Web Servers

Alex Curtiss | CSCI 3403

Patch Notes

Download Python to follow along with live demos:

- www.python.org/downloads/

Will also require installing Flask:

- python -m pip install flask
  (Or `python3 -m pip install flask` depending on aliases)
  (Or just `pip install flask`)

Patch Notes

No recitation this week!

They will start next Friday, September 5.

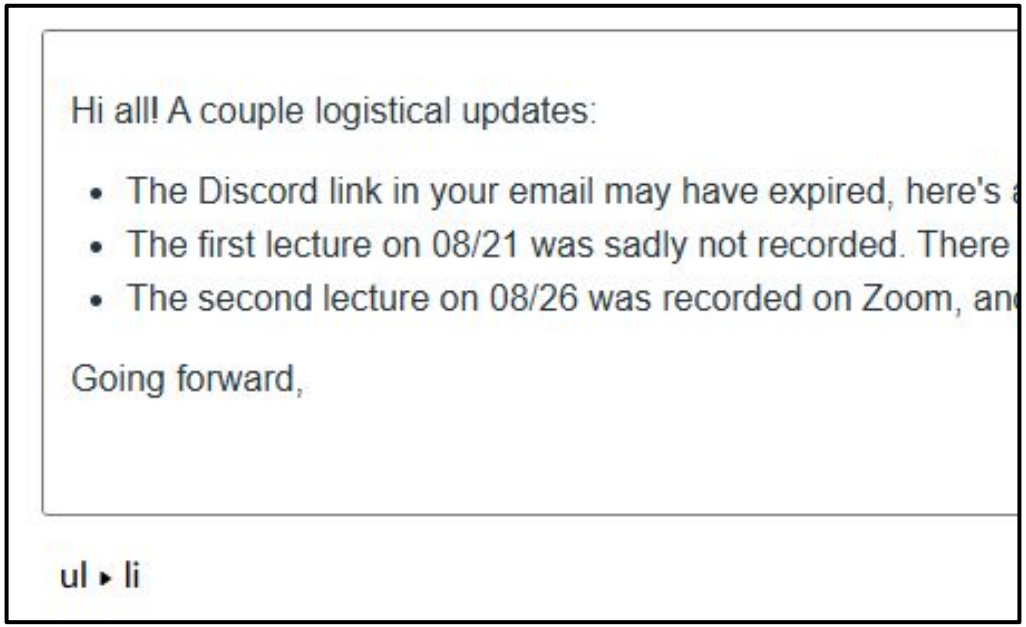
Patch Notes

Various fixes and updates:

- Figured out the recording issues, links to lecture recordings on Canvas (except for the first day, sadly)
- Office hour calendar on Canvas
- Fresh Discord link in syllabus and Canvas announcement
- Expanded the class a little to everyone on the waitlist
- Fixed autograder issues

Patch Notes

HTML tags spotted in the wild on Canvas announcements:



Web Fundamentals

Recap:

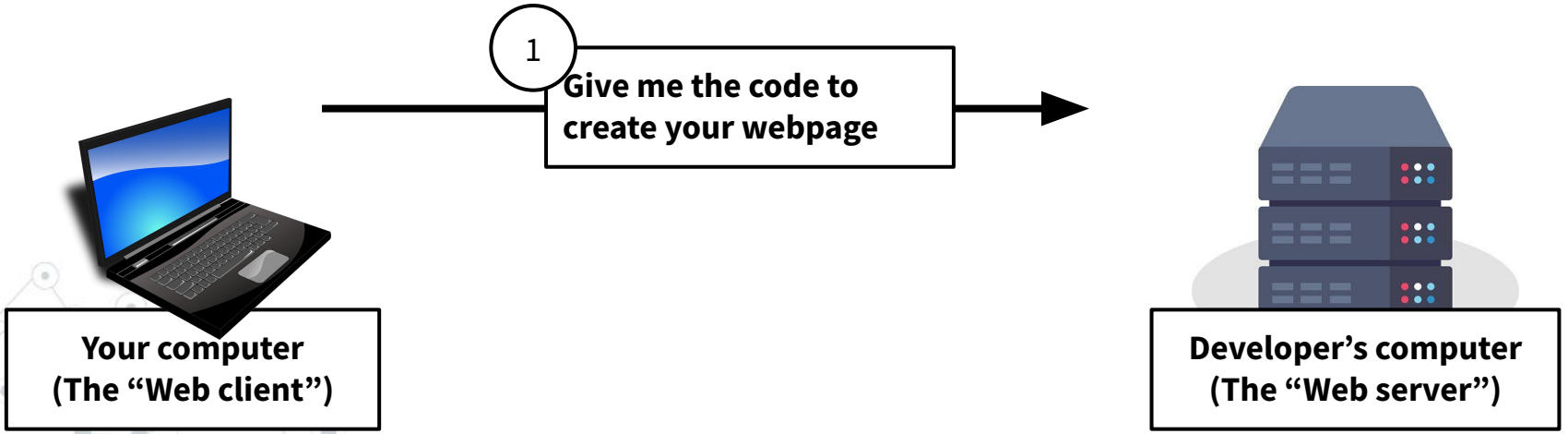
- Client code: Displays the webpage. Written in HTML.
- Server code: Responds to messages from the client.

Your computer
(The “Web client”)

Developer’s computer
(The “Web server”)

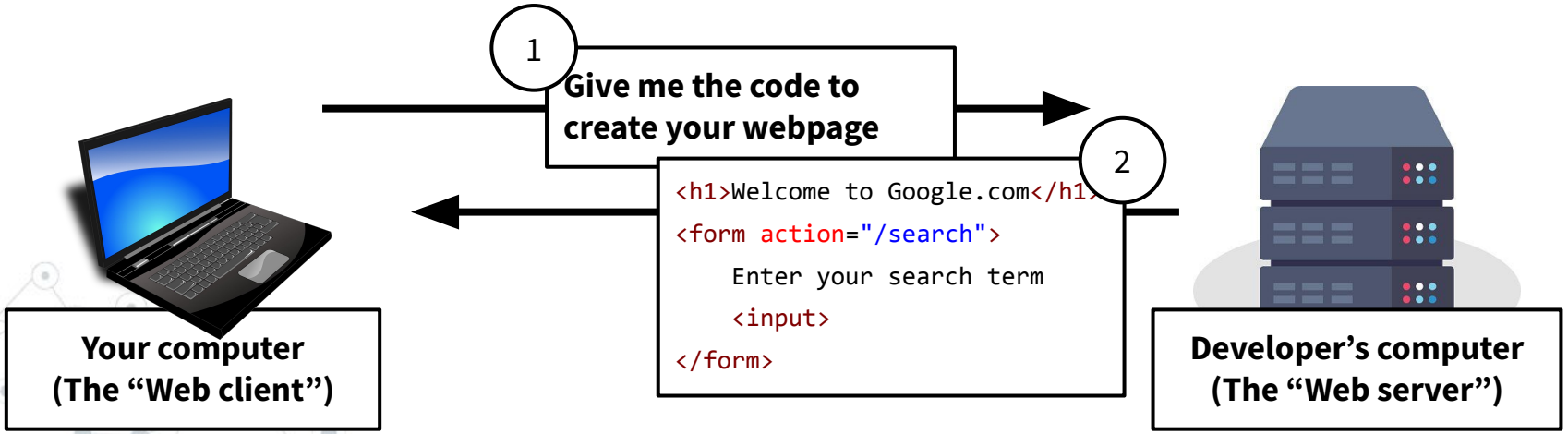
Web Fundamentals

1. Your computer (the “client”) asks for the website from the web developer’s computer (the “server”)



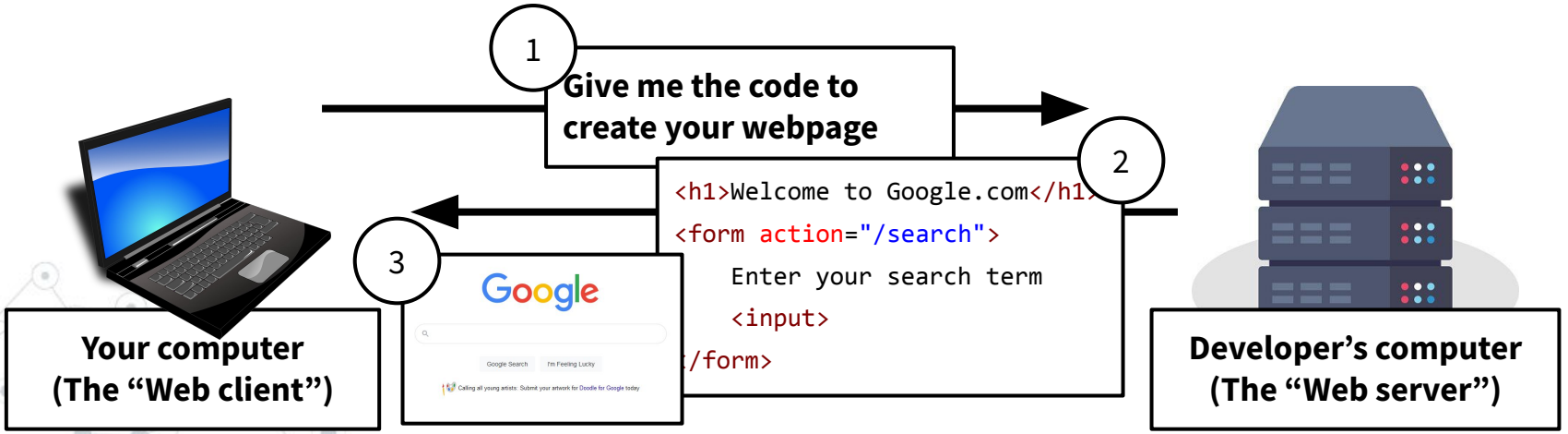
Web Fundamentals

1. Your computer (the “client”) asks for the website from the web developer’s computer (the “server”)
2. The server sends the code needed to create the website



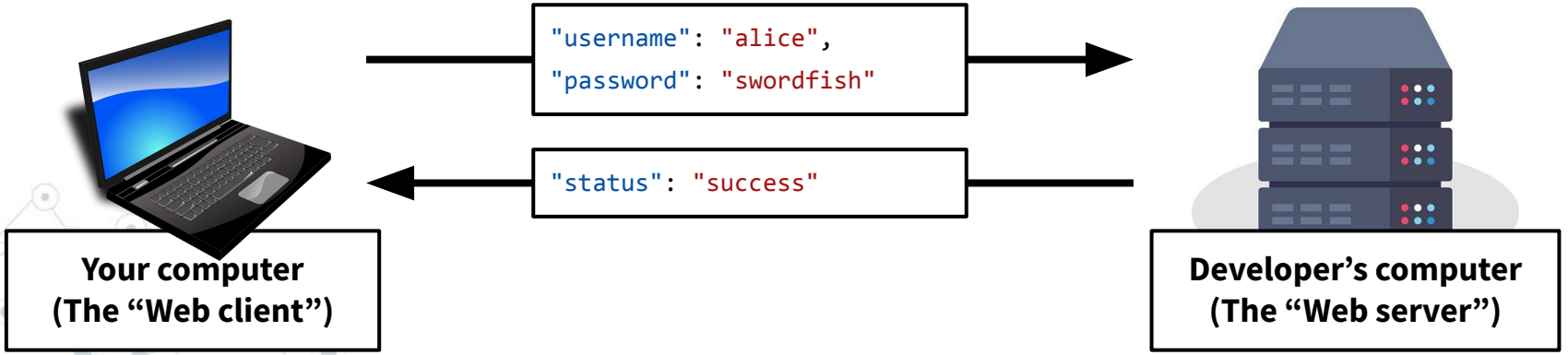
Web Fundamentals

1. Your computer (the “client”) asks for the website from the web developer’s computer (the “server”)
2. The server sends the code needed to create the website
3. The client runs the code to display the website



Web Fundamentals

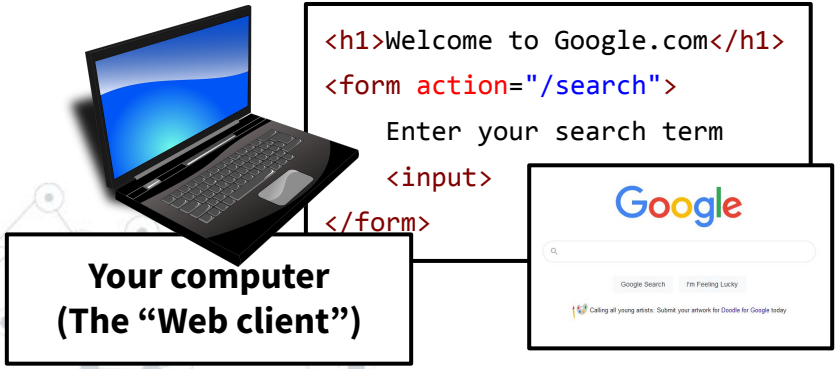
(Optionally: Additional data can be sent back and forth afterwards such as logging in, updating chat messages, etc)

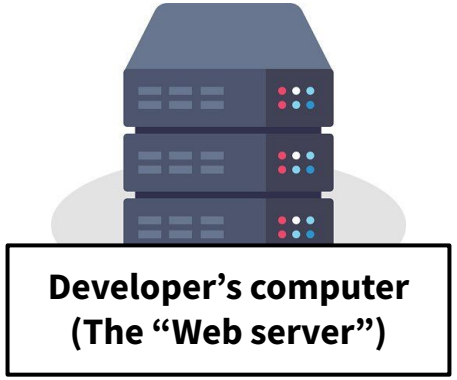


Web Fundamentals

This means web developers write two sets of code: One which runs on the client, and one which runs on the server.

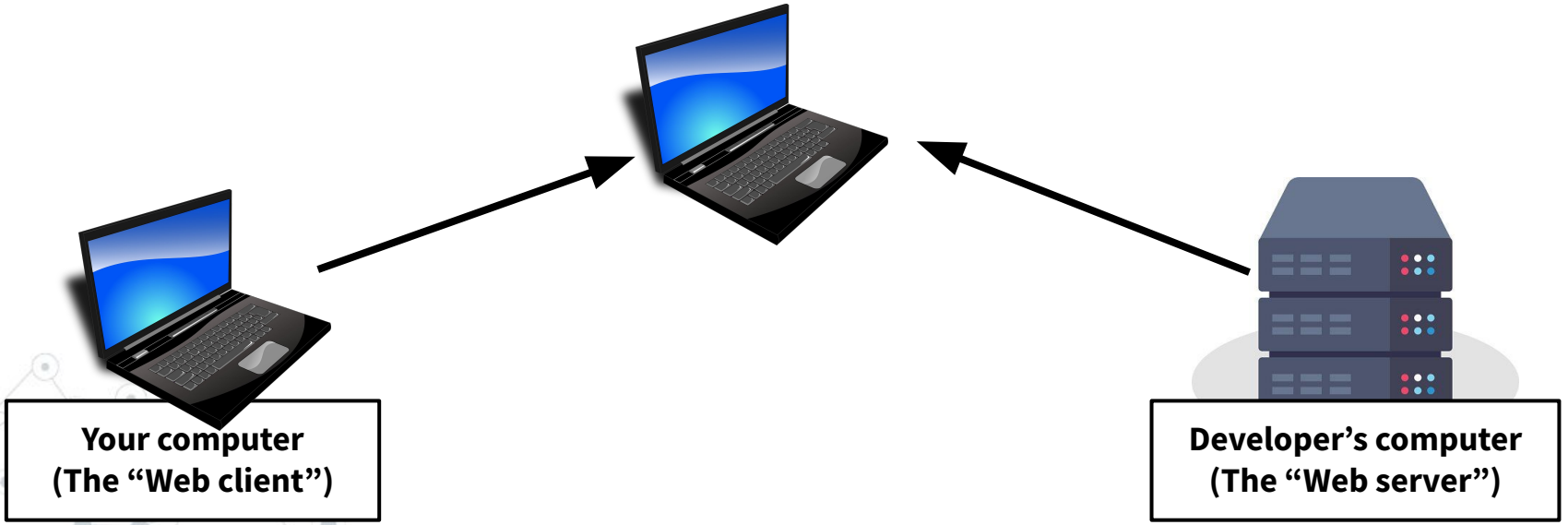
- Client code: Displays the webpage. Written in HTML.
- Server code: Responds to messages from the client.





Server-side code

The client and server are normally different computers, but one computer can act as both for development.



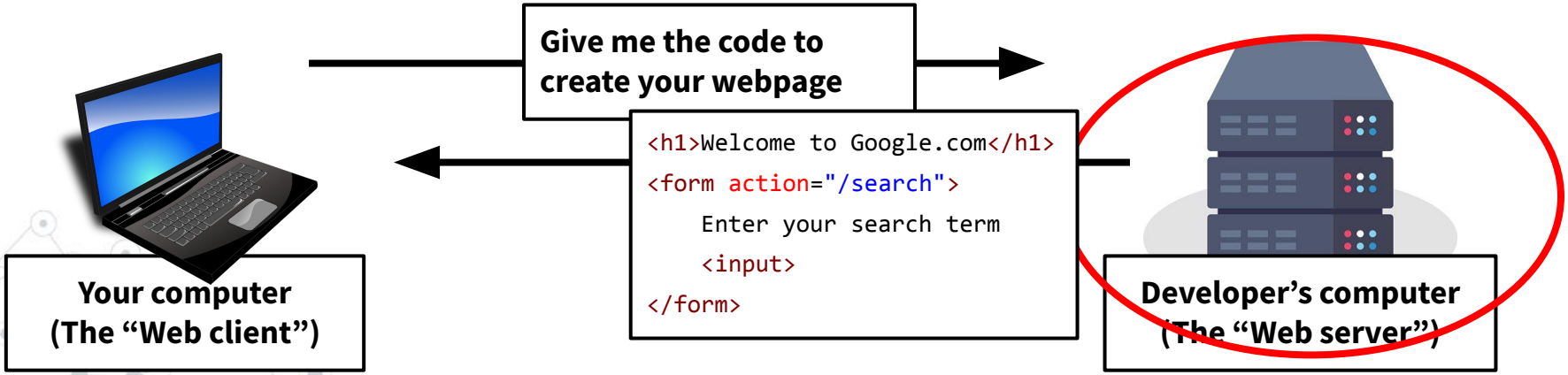
Server-side code

(The rest of this lecture will be a bunch of live demos: the slides have all the necessary information but I would recommend watching the recording)

Server-side code

Server-side code: Sends code or data to the clients when they request it. Constantly listens for new connections.

- Can be written in any language (in this class, Python)



Python

Run a Python file with the command:

```
$ python file.py
```

Python

Python in three bullet points:

- General-purpose programming language like C++
- No brackets, instead whitespace matters
- Types are implicit

```
password = input()
if password == "swordfish":
    return "Correct password!"
else:
    return "Login failed :("
```

Python

Flask: A very simple web server library for Python.

- Installed with this command-line command:
  `python -m pip install flask`

- Imported in a Python file with:

```
from flask import Flask
```

[Demo: Simple Server]

For people online:

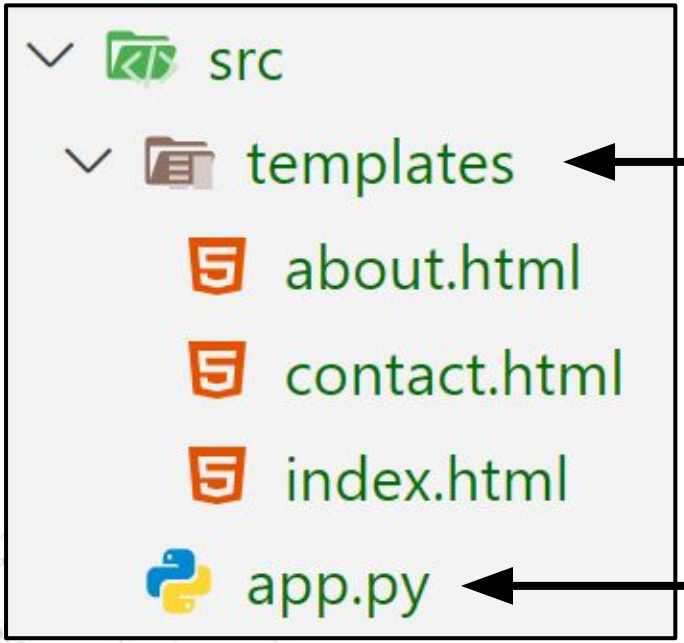
Server-side code

Simple server demo code:

```
from flask import Flask, render_template
app = Flask(__name__)

@app.route("/")
def index():
    # Shorthand for `return open("templates/index.html").read()`
    return render_template("index.html")

app.run()
```

Server-side code

Default file structure, which we will follow for simplicity:



HTML code for clients in **templates/**

Code for the server in **app.py**

Server-side code

You can access a local Flask web server by going to http://localhost:5000.

The “5000” at the end is called a “port number”, and it lets you run multiple web servers on the same computer. The default is 5000, but you can specify it when running Flask:

```
app.run(port=8000)
```

[Demo: Visitor counter]

Server-side code

Flask can dynamically update the page based on variables passed into render_template:

```
return render_template("post.html", like_count=5)
```

Which would change the HTML:

```
<p>Your post has {{ like_count }} likes!</p>
```

Server-side code

Visitor count demo code:

```python
# app.py
from flask import Flask, render_template
app = Flask(__name__)
visitor_count = 0

@app.route("/")
def index():
    visitor_count += 1
    return render_template("index.html",
        visitor_count=visitor_count)

app.run()
```

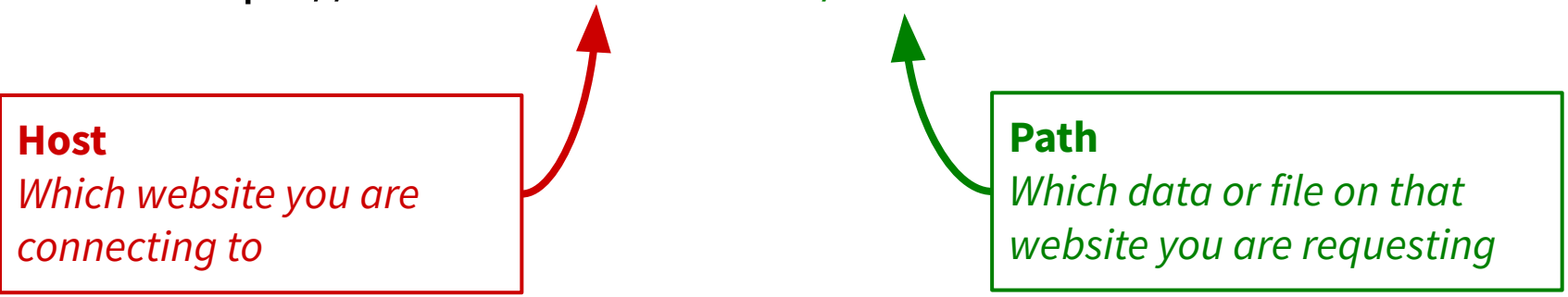
```html
<!-- index.html -->
<h1>Example Site</h1>

<p>
    This site has been visited {{ visitor_count
}} times!
</p>
```

[Demo: Multiple paths]

Server-side code

URLs: Specify which data or file the client is requesting

https://www.colorado.edu/about



Server-side code

Multiple paths demo code:

```
from flask import Flask, render_template
app = Flask(__name__)

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

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

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

app.run()
```

[Demo: Forms]

Server-side code

Request types:

- GET requests ask for code **from** the server
  (this is the default)
- POST requests send data **to** the server

Technically there are more types and other ways to use GET and POST, but for simplicity we will ignore them.

Server-side code

The HTML <form> tag can send data back to the server:

- `<form>` specifies the path *(action)* and whether it is GET or POST *(method)*
- `<input>` tags specify which data is sent

```
<form action="/login" method="post">
    <input type="text" name="username">
    <input type="password" name="password">
</form>
```

Server-side code

On the server:

- Form inputs are stored in “request.form” based on the “name” attribute of the <input> tags:

```
@app.route("/login", methods=["POST"])
def login_post():
    username = request.form["username"]
    password = request.form["password"]
    if username == "alex" and password == "swordfish":
        return "Correct!"
```

[Demo: Forms]

Recap

- **Web servers**
- **Python and Flask basics**
- **URL concepts:**
  - Ports
  - Paths
  - Methods (GET, POST)

Next lecture: Headers and cookies

Server-side code

Form demo code:

```python
from flask import Flask, render_template, request, redirect

app = Flask(__name__)

messages = []

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

@app.route("/message", methods=["post"])
def message():
    message = request.form["message"]

    messages.append(message)

    if len(messages) > 20:
        messages.pop(0)

    return redirect("/")

app.run()
```

```html
<!-- index.html -->
<pre id="chat">
    {{ messages }}
</pre>

<form action="/post" method="post">
    <input type="text" name="message">
    <input type="submit" value="Submit">
</form>
```