CS 4220

- Current Trends in Web Design & Development -

Cydney Auman

AGENDA

What is a Server?

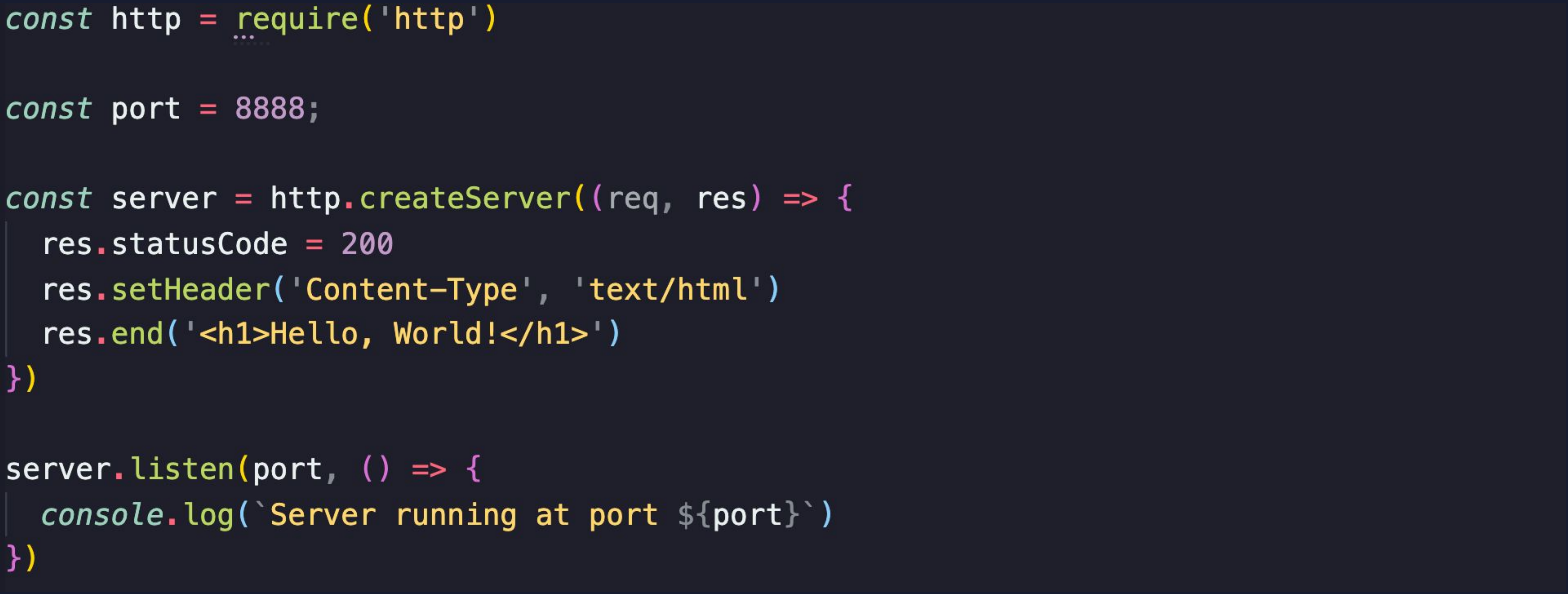
A **server** is a computer or a device that provides some functionality for other programs or devices, called "clients".

This architecture is called the client–server model. Servers provide various functionalities, often called "services", such as sharing data or resources. An API is an example of one such service Servers can provide.

A single server can serve multiple clients, and a single client can use and access multiple servers.

Node.js Server

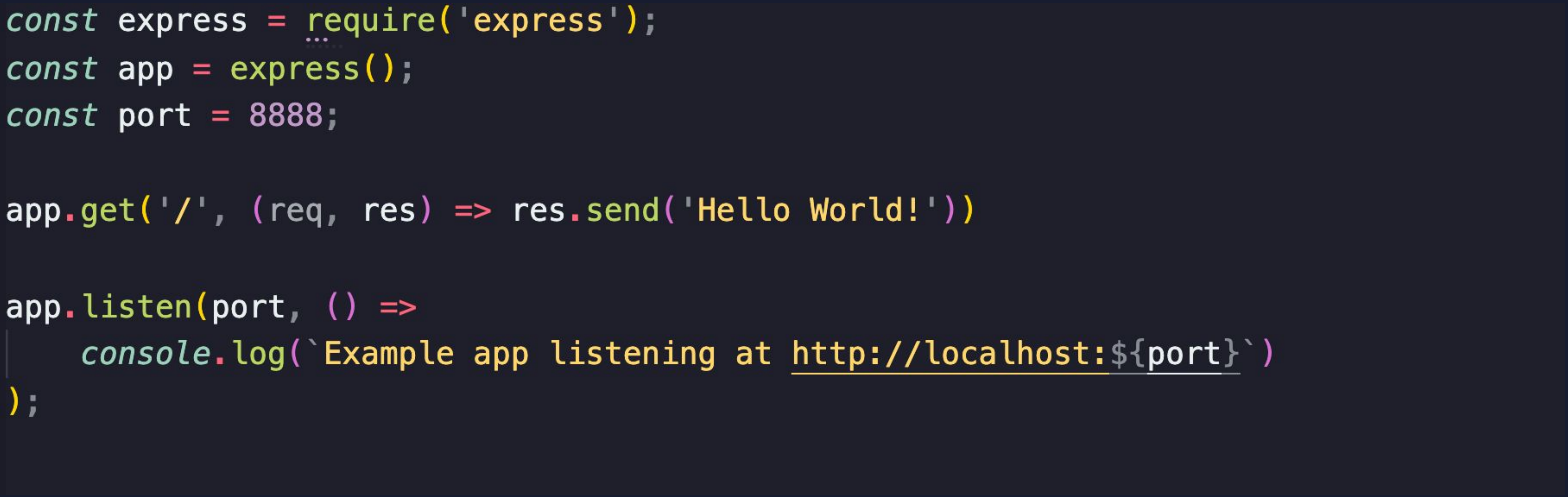
Using the http module and then the `createServer()` function Node.js makes it simple to spin up a server.



```javascript
const http = require('http')

const port = 8888;

const server = http.createServer((req, res) => {
  res.statusCode = 200
  res.setHeader('Content-Type', 'text/html')
  res.end('<h1>Hello, World!</h1>')
})

server.listen(port, () => {
  console.log(`Server running at port ${port}`)
})
```

Node Express.js Server

Using the **Express** module and the same Node.js server can be created in a cleaner and simpler manner. Express.js like many Node Modules enhance the capabilities of Node.js.



```
const express = require('express');
const app = express();
const port = 8888;

app.get('/', (req, res) => res.send('Hello World!'))

app.listen(port, () =>
    console.log(`Example app listening at http://localhost:${port}`)
);
```

Why Express.js ?

Express.js describes itself as a "a minimal and flexible Node.js web application framework, providing a robust set of features for building single and multi-page, and hybrid web applications."

In short, it's a framework for building web applications with Node.js.   And Express.js like many other Node Modules enhance the capabilities of Node.js.

A Closer Look At Servers

```
// require in the express module
const express = require('express');

// call the express function which creates the app with a lot of features and functionality
const app = express();

// define the port we want the server to listen on (common ports are 8000 and 8080 and 8888)
const port = 8888;

// use the app to the define a route - this particular route is the base route of /
app.get('/', (request, response) => {
   // respond to any clients with a simple plain text
   response.send('Hello World!');
});

// spins up the server on the defined port. once the server is alive it will log
app.listen(port, () =>
   console.log(`Example app listening at http://localhost:${port}`)
);
```

Request

request is a request that comes from the client such as a browser or CLI - this sometimes shortened to **req**.

The request argument contains information about the request, such as its url, http method, headers, body and etc.

```
app.get('/', (request, response) => {
    // respond to any clients with a simple plain text
    response.send('Hello World!');
});
```

Response

The **response** is the next argument in the function. Just like the prior argument is often shortened to **res**.

With each response, you get the data ready to send, and then you call `response.send()` or `response.json()`. Eventually, you must call this method. This method does the actual sending of data. If this method is not called, the server request just hangs forever.

```
app.get('/', (request, response) => {
   // respond to any clients with a simple plain text
   response.send('Hello World!');
});
```

Middleware in express.js

Middleware consists of functions that are invoked by the express.js routing layer before the final request handler. It sits in the middle between a raw request and the final intended route.

These functions have access to the request object (req) and the response object (res). Since they are always invoked in the order they are added, they are treated like a stack.

Middleware functions can perform the following tasks:

- Execute any code.
- Make changes to the request and the response objects.
- End the request-response cycle.
- Call the next middleware function in the stack.

Middleware in express.js

```javascript
const express = require('express');
const bodyParser = require('body-parser');

const app = express();
const port = 8888;

// attempt to parse incoming request bodies before the request handlers
// then the JSON is available under req.body
app.use(bodyParser.json({ type: 'application/json' }));
```

```
console.log(‘Week 12’);
console.log(‘Code Examples’);
```

Lab, Homework and Prep

Lab Time

- Run the Demo Examples
- Work on the Extra Credit Assignment

Preparation for Next Week

- Readings:

  Eloquent JavaScript - Chapter 13

  https://vuejs.org/v2/guide/