Chapter 3
Node.js

- Introduction
- File System Package
- Blocking vs Non-Blocking I/O
- Http Package

Introduction to Node.js

What is Node.js?

- *JavaScript runtime environment*
- *Single-threaded, based on event-driven non-blocking I/O model*
- *Open source and cross-platform*

Introduction to Node.js

What can Node.js used for?

- *Server-side applications*
- *Microservices and API*
- *Generate dynamic page content*
- *Read and write files on the file system*
- *Read and write on the database*

Don’t use Node.js for apps with heavy server-side processing

Introduction to Node.js

Node.js is

- *Single-threaded*
- *Event-driven*
- *non-blocking I/O*

Node.js = can handle large numbers of simultaneous connections without blocking or slowing down.

Introduction to Node.js

Blocking I/O model: *a server will wait for a request to be completed before moving on to the next request.*

Non-blocking I/O model: *Node.js uses an event-driven approach where each request is treated as an event, and the server responds to each event as it occurs.*

Node.js = When a request is received, it registers a callback function to be executed when the request is complete. The server then continues processing other.

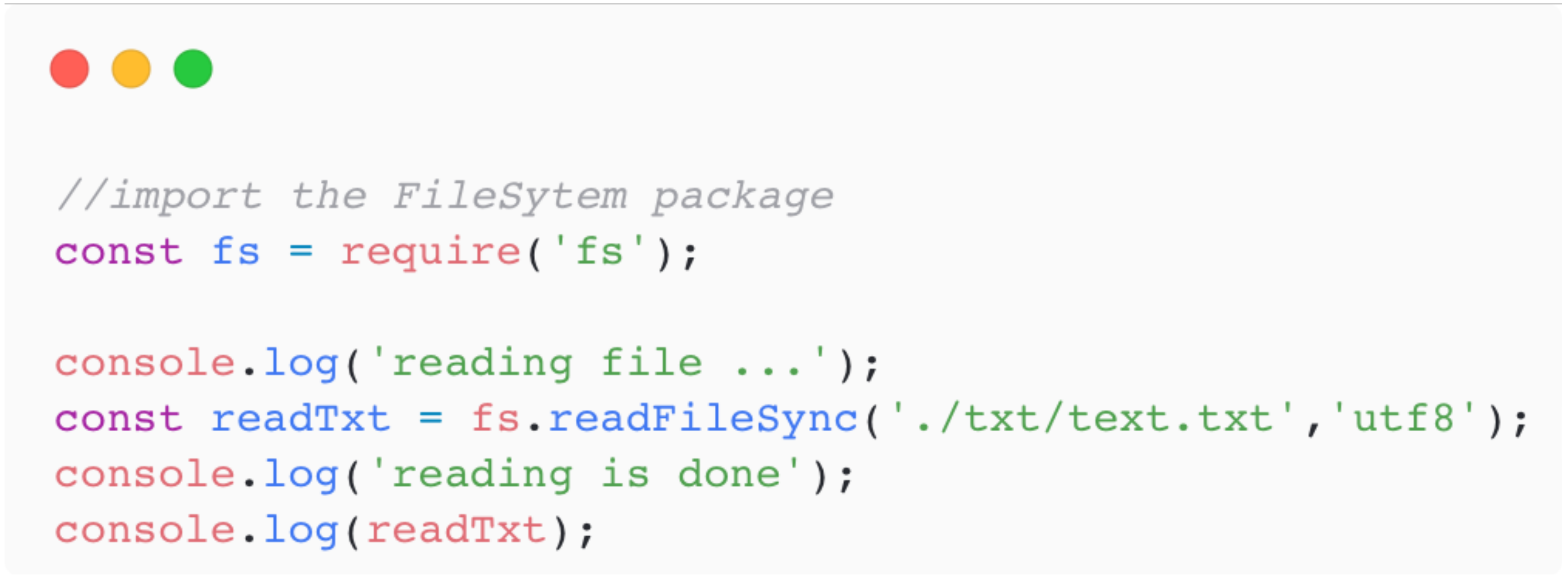
Introduction to Node.js

- *Node.js also has a vast library of modules that can be easily installed and integrated into Node.js applications using the* ***npm*** *(Node Package Manager) tool.*

- *This makes it easy for developers to quickly build powerful applications without having to write everything from scratch.*

File System Package

- *Read data from a file*

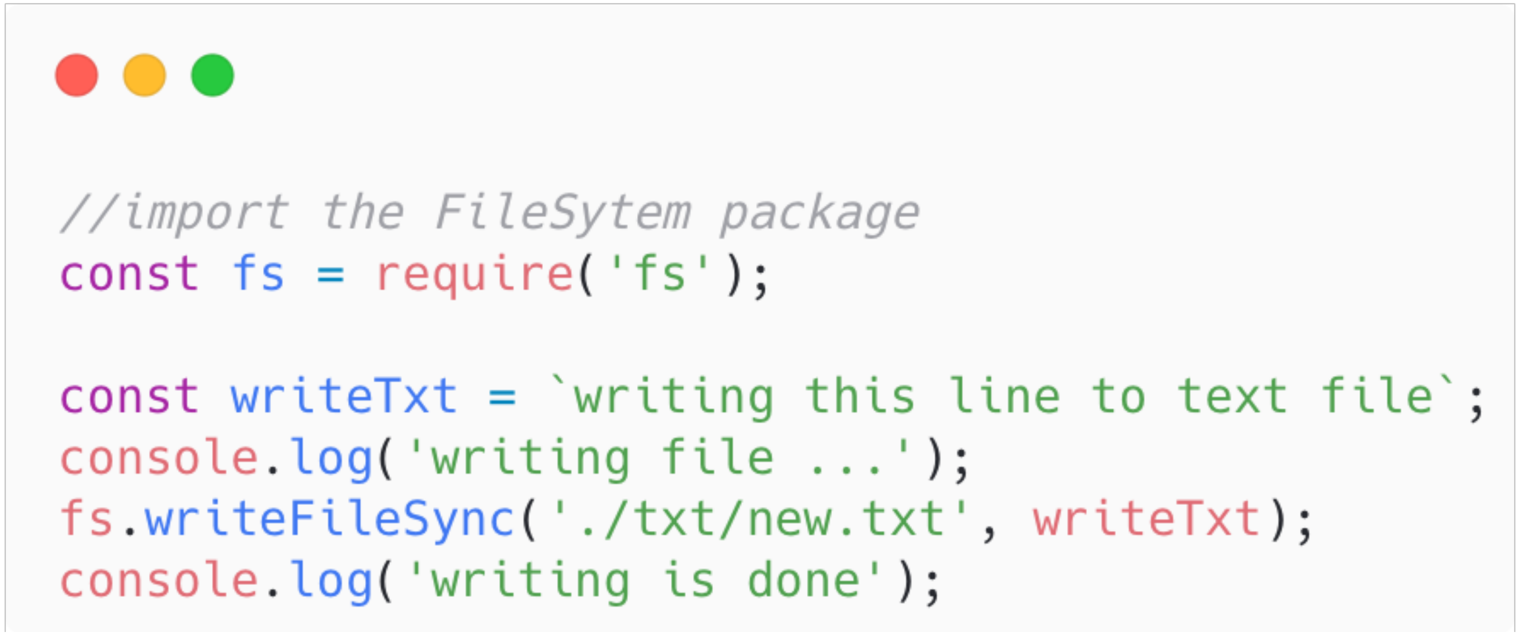


```
//import the FileSytem package
const fs = require('fs');

console.log('reading file ...');
const readTxt = fs.readFileSync('./txt/text.txt','utf8');
console.log('reading is done');
console.log(readTxt);
```

File System Package

- *Writing data from a file*



```
//import the FileSytem package
const fs = require('fs');

const writeTxt = `writing this line to text file`;
console.log('writing file ...');
fs.writeFileSync('./txt/new.txt', writeTxt);
console.log('writing is done');
```

Blocking vs non-blocking

Synchronous code

- *fs.readFileSync is a* ***synchronous*** *function*
- *In a synchronous code, each statement is processed line by line*
- *Each line waits for the results of the previous line*
- *Cons: slow operations will cause the processing to be blocked*

Synchronous code = blocking code

Node.js is designed to be non-blocking

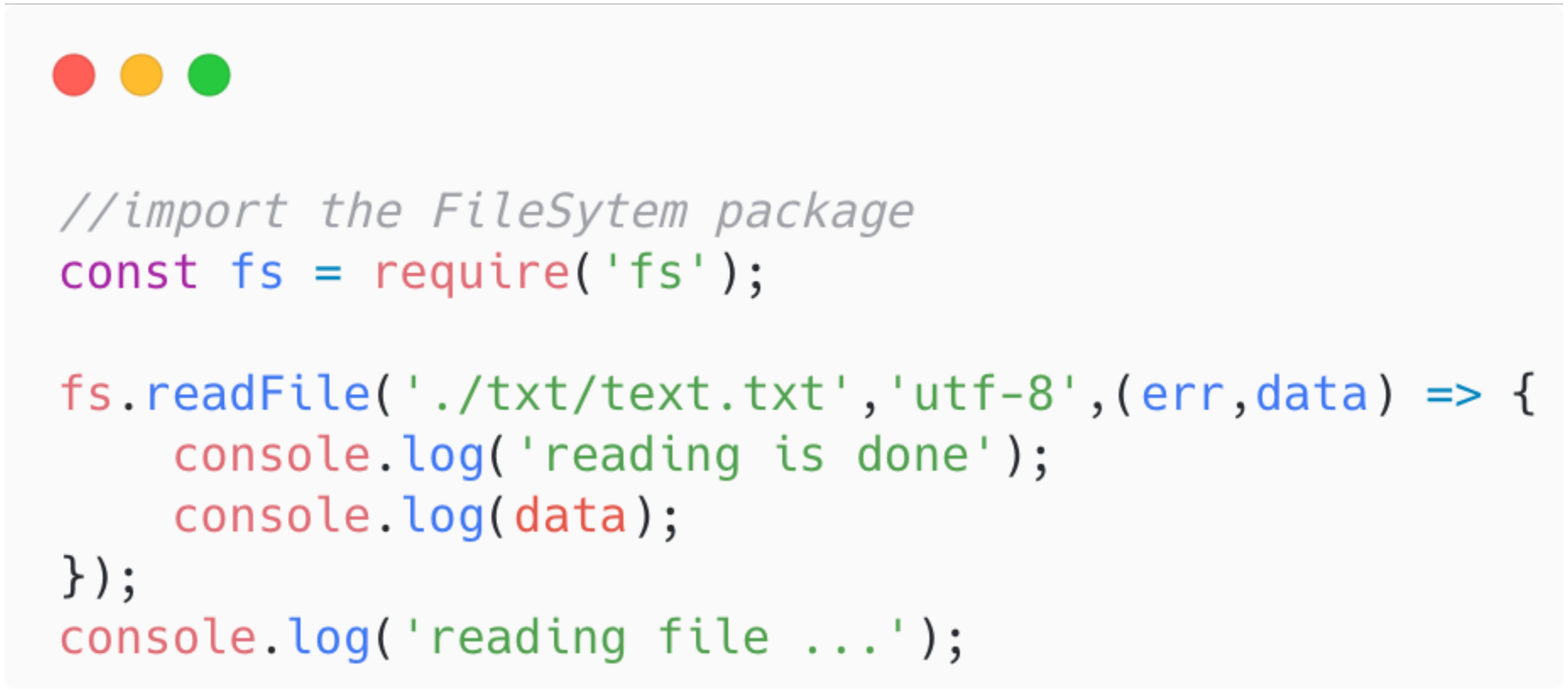
Blocking vs non-blocking

Asynchronous code/non-blocking

- *fs.readFile is an a**synchronous** function*
- *In a asynchronous code, we don’t wait for the asynchronous statement to finish*
- *Instead, we register a callback to be called once the asynchronous statement is finished*
- *The callback function handle any result of the asynchronous statement*
- *Pros: slow operations won’t block the code*

Blocking vs non-blocking

Asynchronous code/non-blocking



```javascript
//import the FileSytem package
const fs = require('fs');

fs.readFile('./txt/text.txt','utf-8',(err,data) => {
    console.log('reading is done');
    console.log(data);
});
console.log('reading file ...');
```

Blocking vs non-blocking

*Asynchronous programming is a programming technique that allows a program to perform tasks in the background **without blocking** the main thread.*

JavaScript uses a callback-based model for asynchronous programming

***Asynchronous functions** will typically take a callback function as a parameter that will be called once the operation is complete.*

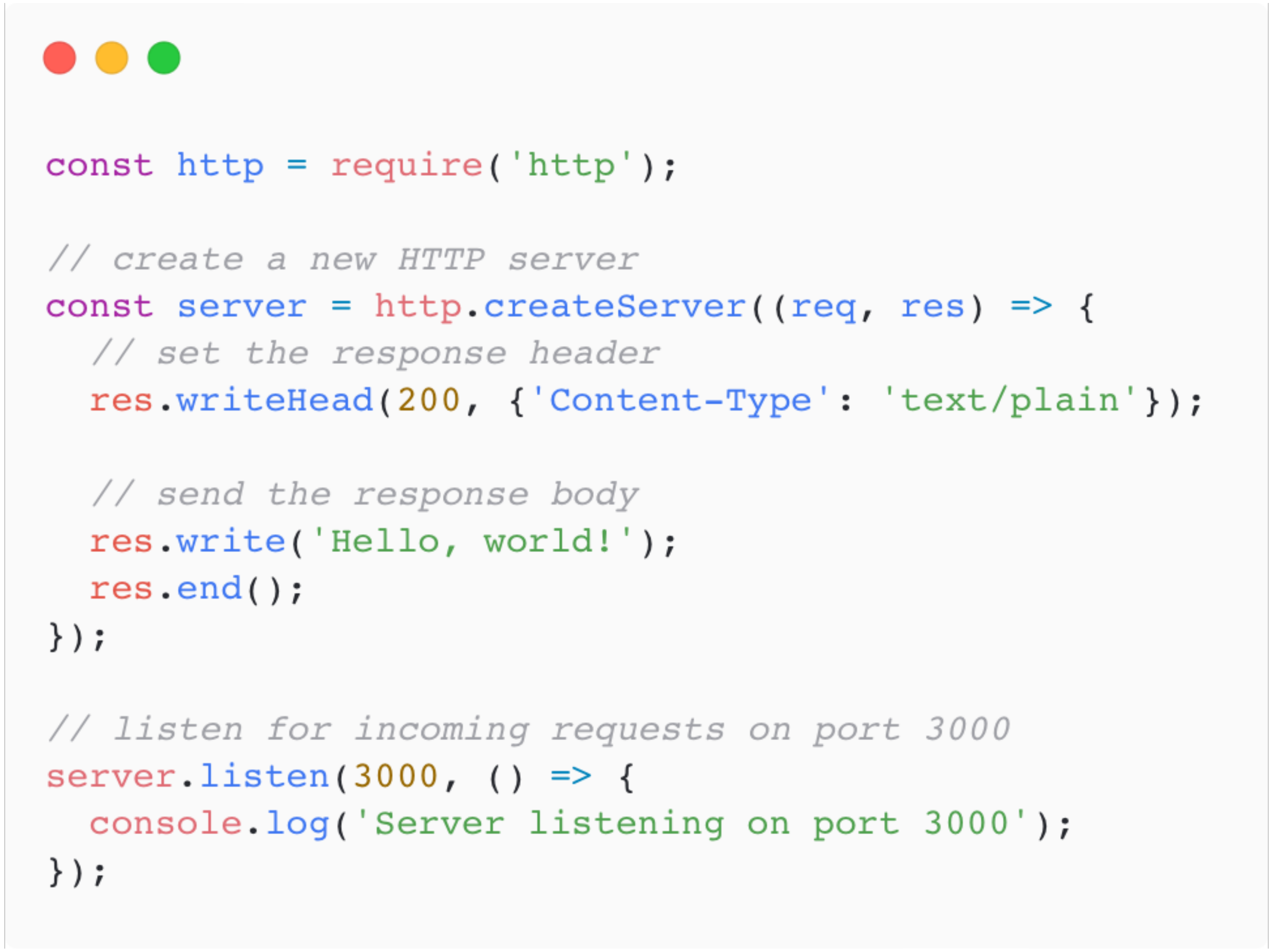
Http Package

HTTP package provides **functionality for working** with the Hypertext Transfer Protocol (HTTP)

It allows developers to **create HTTP servers**, and provides a range of features for handling requests and responses.

Http Package

How to create a server using HTTP?



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

// create a new HTTP server
const server = http.createServer((req, res) => {
  // set the response header
  res.writeHead(200, {'Content-Type': 'text/plain'});

  // send the response body
  res.write('Hello, world!');
  res.end();
});

// listen for incoming requests on port 3000
server.listen(3000, () => {
  console.log('Server listening on port 3000');
});
```

HTTP Response

HTTP response Once the server receives a request, it responds with a response that contains:

- Status code
- Headers
- Body

HTTP Response

Status Code:

The status Code is a three-digit number that the HTTP server sends in the response to ***indicate the status*** of the request.

The status code is included in the HTTP response header and is used by the client to determine how to ***interpret*** the response.

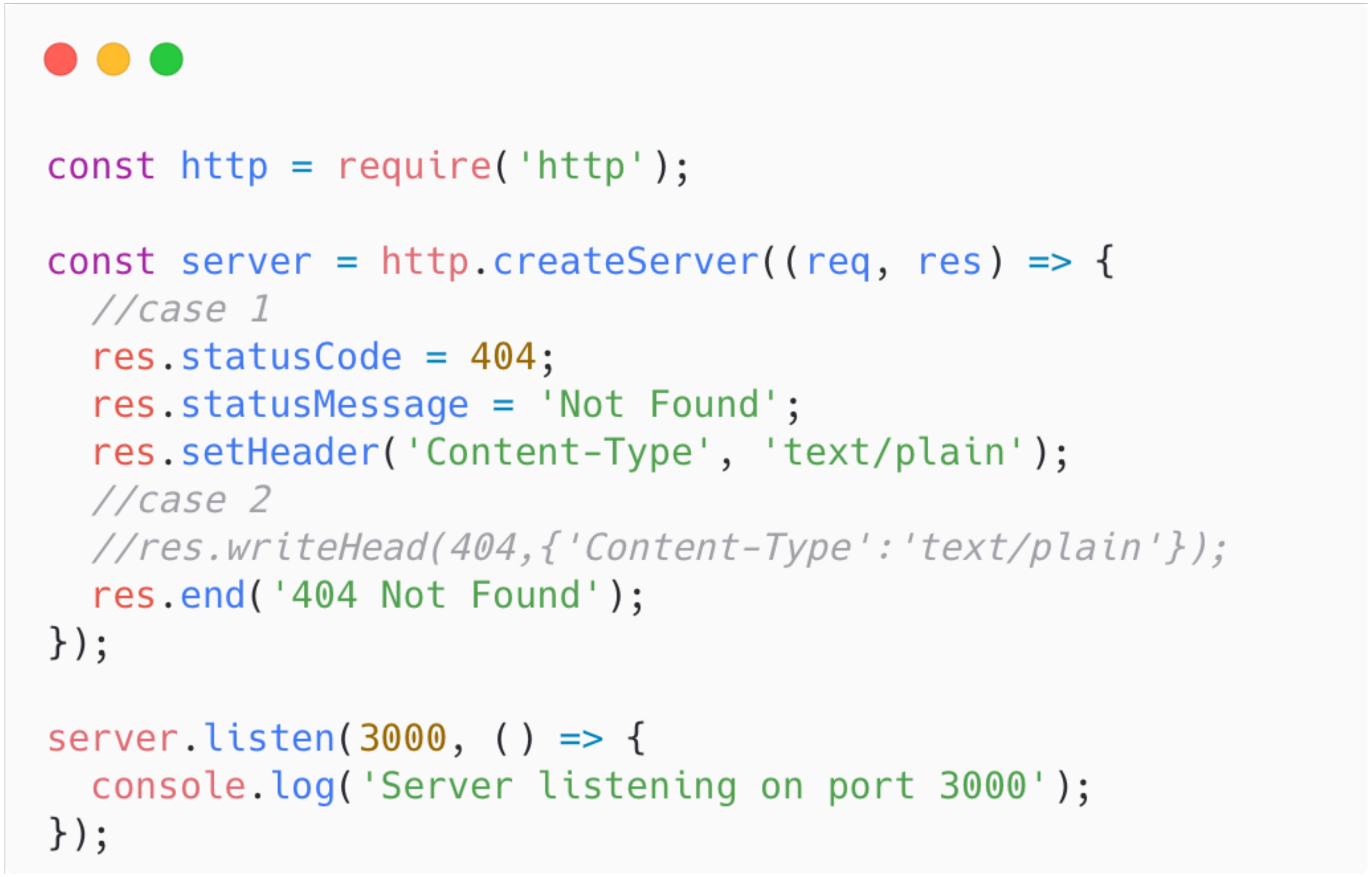
HTTP Response

Status Code: examples

- ***200 OK:*** The request was successful and the server is returning the requested data.
- ***201 Created:*** The request was successful and a new resource has been created on the server.
- ***204 No Content:*** The request was successful but there is no data to return in the response.
- ***400 Bad Request:*** The request was malformed or invalid.
- ***401 Unauthorized:*** The request requires authentication, but the client has not provided valid credentials.
- ***403 Forbidden:*** The request is valid, but the server is refusing to fulfill it.
- ***404 Not Found:*** The requested resource could not be found on the server.
- ***500 Internal Server Error:*** An error occurred on the server while processing the request.
- ***503 Service Unavailable:*** The server is currently unable to handle the request due to maintenance or overload.

HTTP Response

Status Code: Syntax



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

const server = http.createServer((req, res) => {
  //case 1
  res.statusCode = 404;
  res.statusMessage = 'Not Found';
  res.setHeader('Content-Type', 'text/plain');
  //case 2
  //res.writeHead(404,{'Content-Type':'text/plain'});
  res.end('404 Not Found');
});

server.listen(3000, () => {
  console.log('Server listening on port 3000');
});
```

HTTP Response

Headers:

HTTP headers are pieces of ***additional information*** that can be sent along with an HTTP request or response.

They provide metadata about the request or response and help the client and server communicate with each other more effectively.

Examples:

- Content-Type (res/req)
- Accept (req)
- User-Agent (req)
- Authorization (req)
- Cookie (req)
- Set-Cookie (res)
- Date (res)

HTTP Response

Headers/Content-Type: is used to specify the type of the content being sent in a response or in a request.

- ***text/plain:*** Plain text content
- ***text/html:*** HTML content
- ***text/css:*** Cascading Style Sheets (CSS) content
- ***text/javascript:*** JavaScript content
- ***application/json:*** JSON content
- ***application/pdf:*** Portable Document Format (PDF) content
- ***image/jpeg:*** JPEG image content
- ***image/png:*** PNG image content
- ***audio/mpeg:*** MP3 audio content
- ***video/mp4:*** MP4 video content

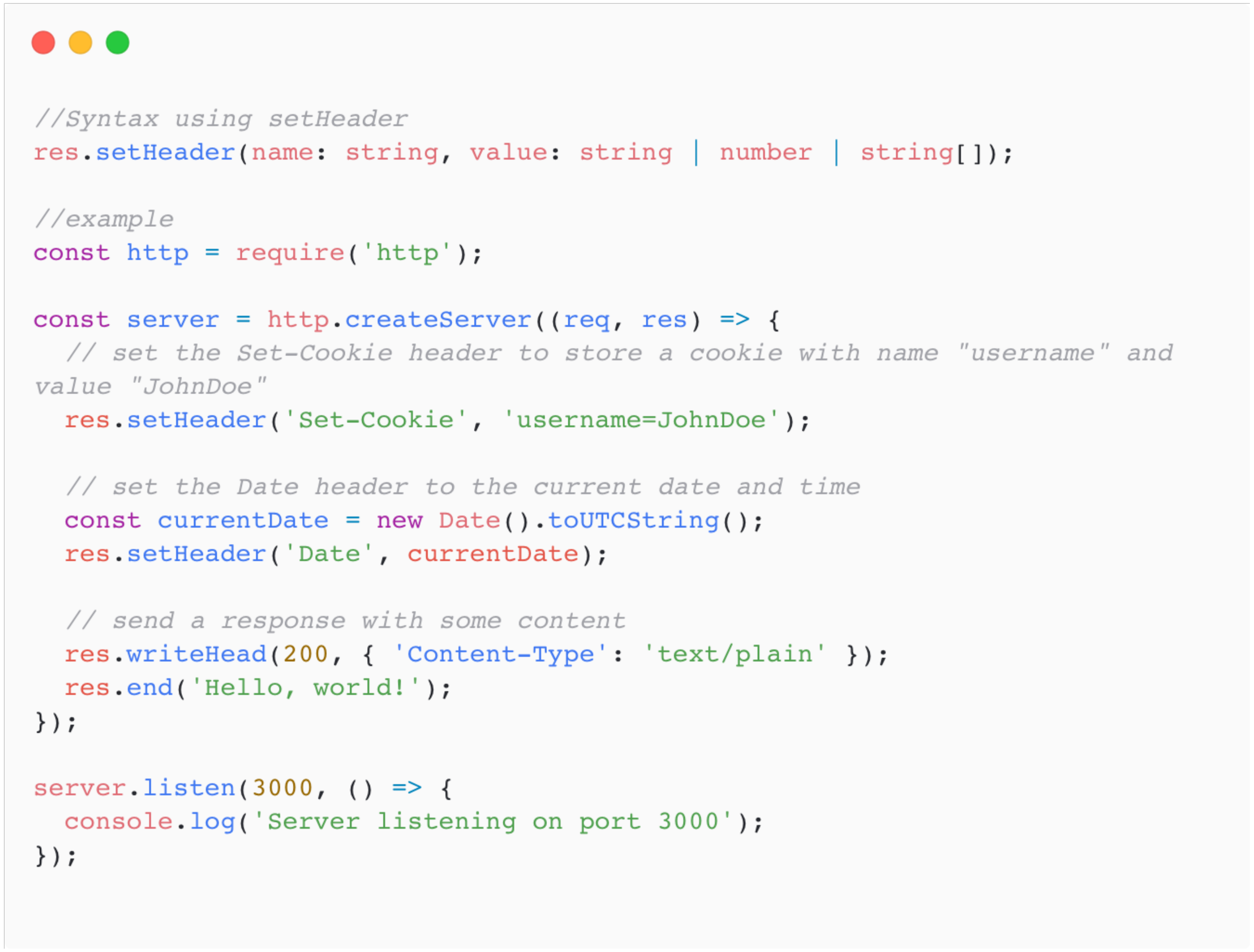
HTTP Response

HTTP Request/Headers/Accept: is used to specify the type of the content the client can handle in the response.

- ***Accept: text/html:*** indicates that the client can handle HTML-formatted content in the response.
- ***Accept: application/json:*** indicates that the client can handle JSON-formatted content in the response.
- ***Accept: text/*:*** indicates that the client can handle any type of text-based content in the response.
- ***Accept: */*:*** indicates that the client can handle any type of content in the response.

Http Package

HTTP Response/Headers



```
//Syntax using setHeader
res.setHeader(name: string, value: string | number | string[]);

//example
const http = require('http');

const server = http.createServer((req, res) => {
  // set the Set-Cookie header to store a cookie with name "username" and
value "JohnDoe"
  res.setHeader('Set-Cookie', 'username=JohnDoe');

  // set the Date header to the current date and time
  const currentDate = new Date().toUTCString();
  res.setHeader('Date', currentDate);

  // send a response with some content
  res.writeHead(200, { 'Content-Type': 'text/plain' });
  res.end('Hello, world!');
});

server.listen(3000, () => {
  console.log('Server listening on port 3000');
});
```

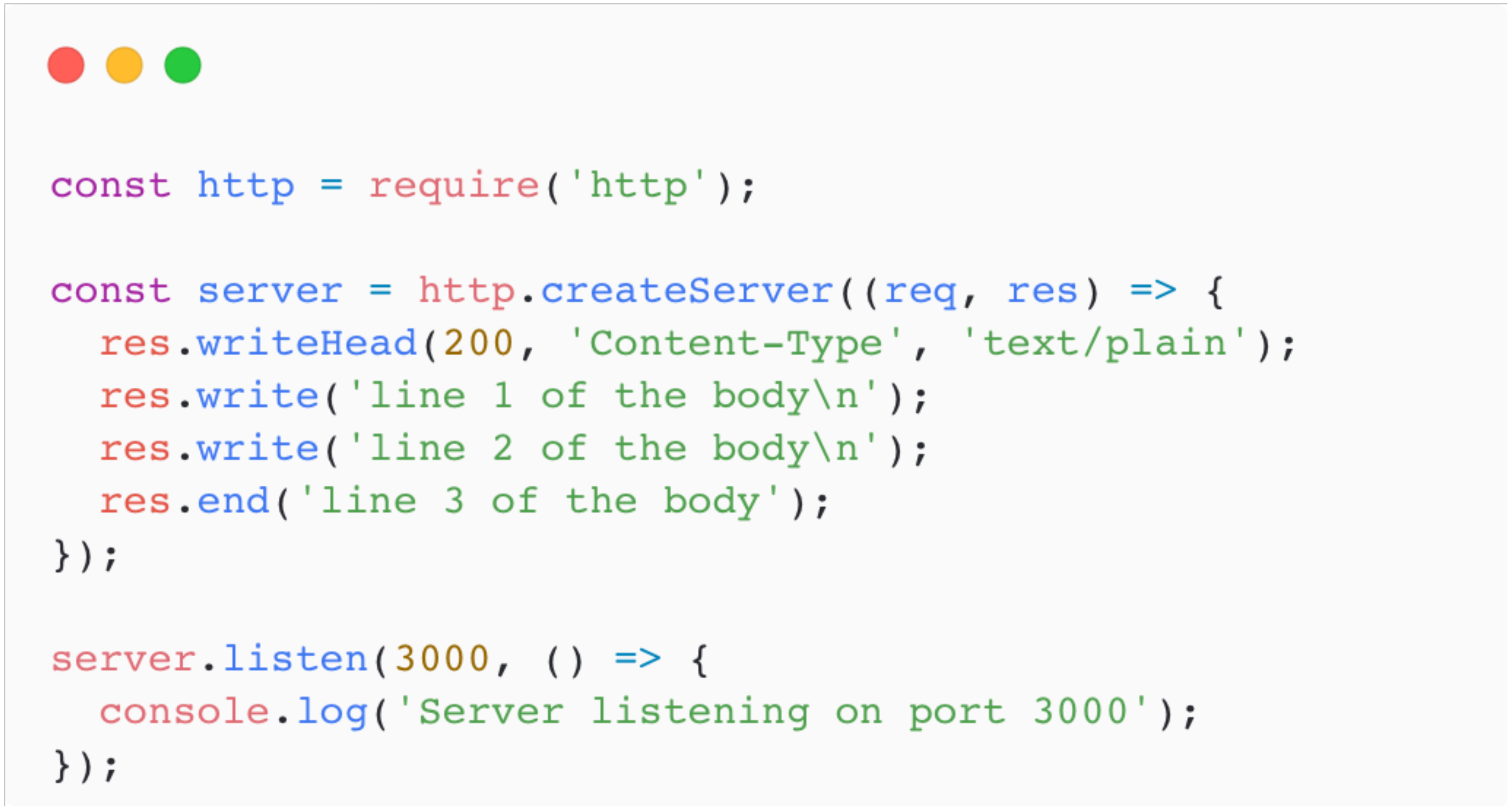
HTTP Response

Body: refers to the main content of the message being sent.

- In an HTTP ***request***, the body is typically used to send data from the client to the server.
- In an HTTP ***response***, the body is used to send data from the server back to the client.

HTTP Response

HTTP Response/Body



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

const server = http.createServer((req, res) => {
  res.writeHead(200, 'Content-Type', 'text/plain');
  res.write('line 1 of the body\n');
  res.write('line 2 of the body\n');
  res.end('line 3 of the body');
});

server.listen(3000, () => {
  console.log('Server listening on port 3000');
});
```

Handle HTTP requests

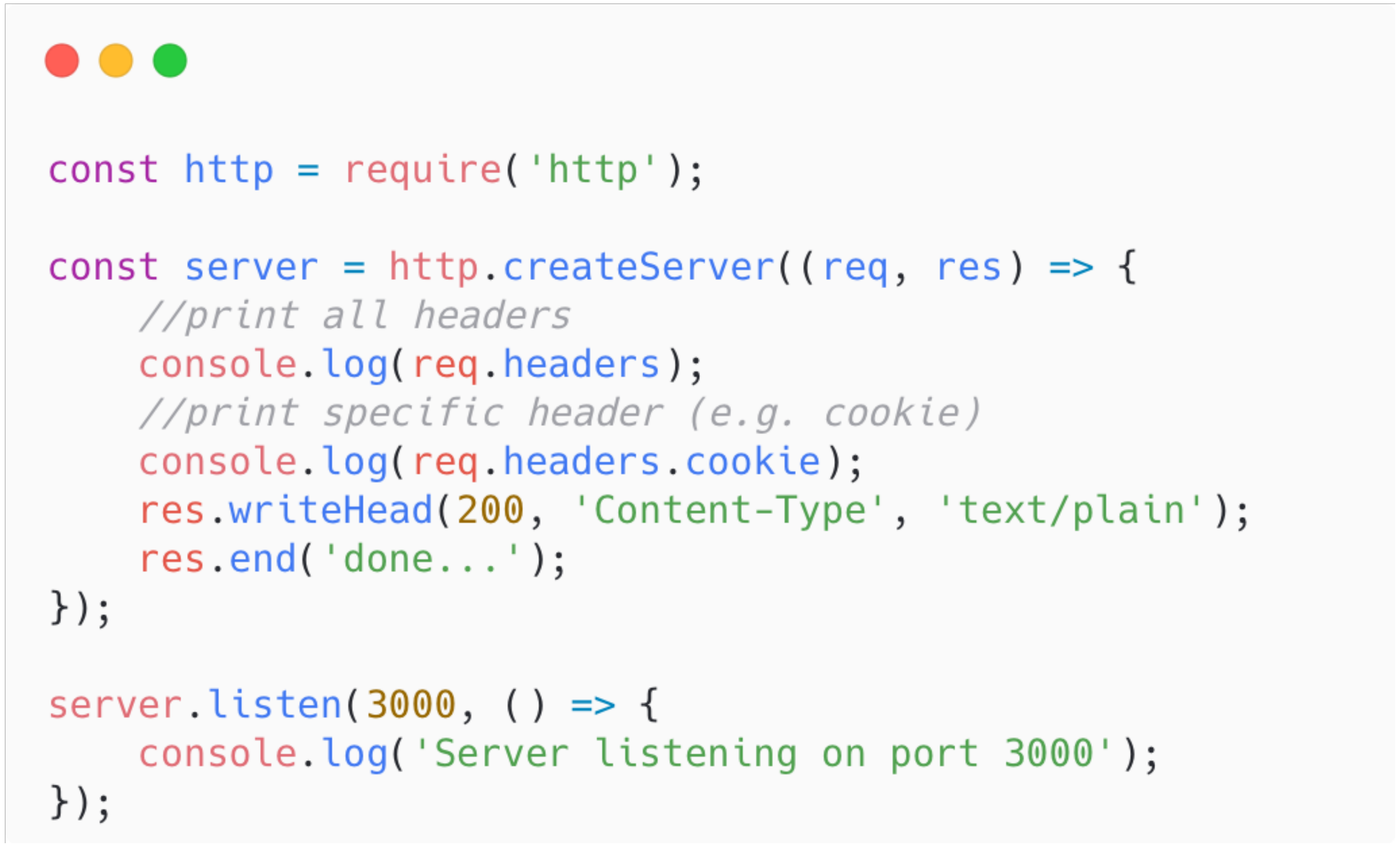
In a Node.js HTTP server, the ***req*** object represents the incoming ***HTTP request*** sent by the client.

The req object contains a lot of information about the request, including:

- req.headers
- req.url
- req.method
- req.params
- req.body
- req.query

Handle HTTP requests

req.headers: *any additional information attached to the request*



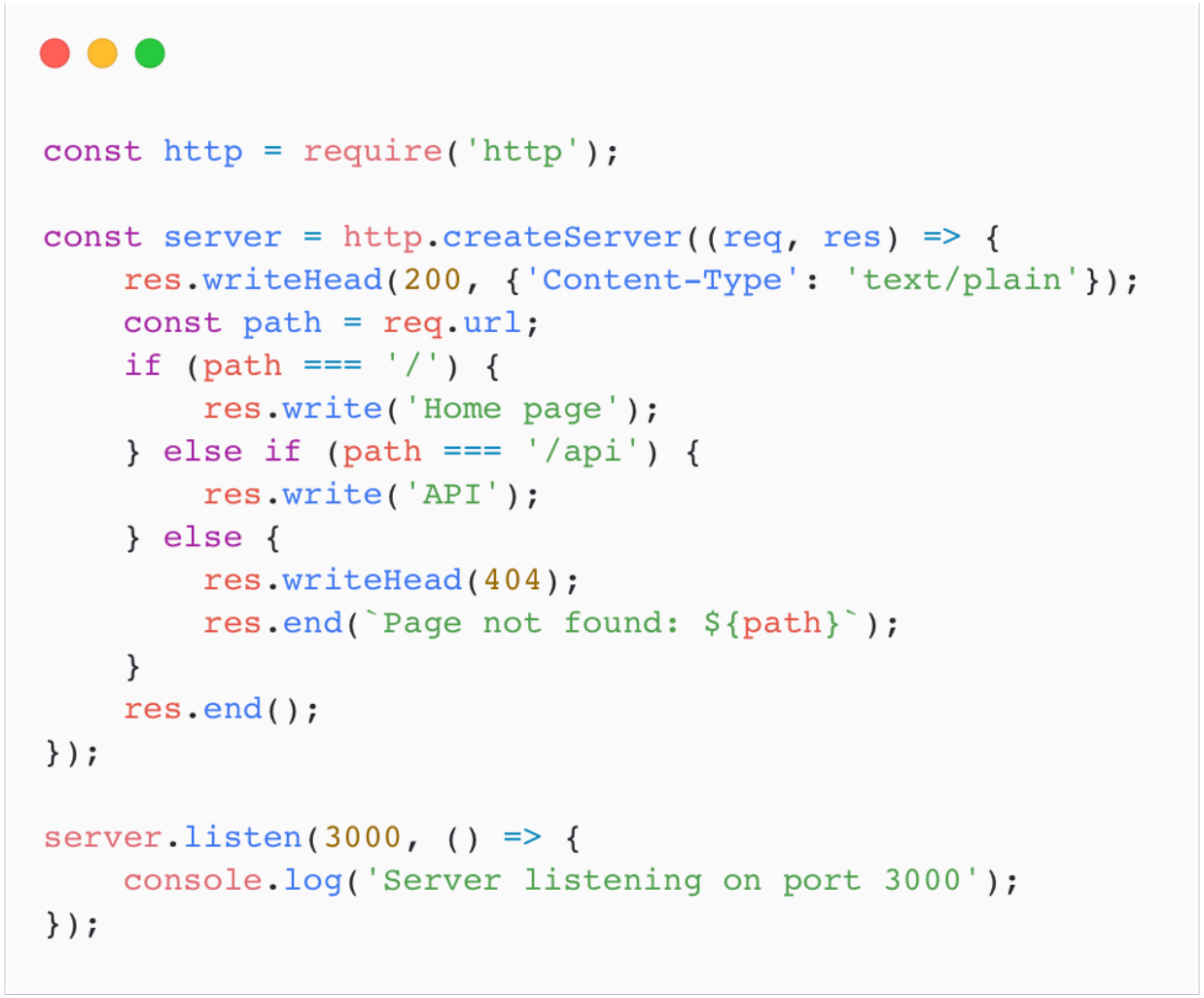
```
const http = require('http');

const server = http.createServer((req, res) => {
    //print all headers
    console.log(req.headers);
    //print specific header (e.g. cookie)
    console.log(req.headers.cookie);
    res.writeHead(200, 'Content-Type', 'text/plain');
    res.end('done...');
});

server.listen(3000, () => {
    console.log('Server listening on port 3000');
});
```

Handle HTTP requests

req.url: *the requested URL (including any query string parameters)*



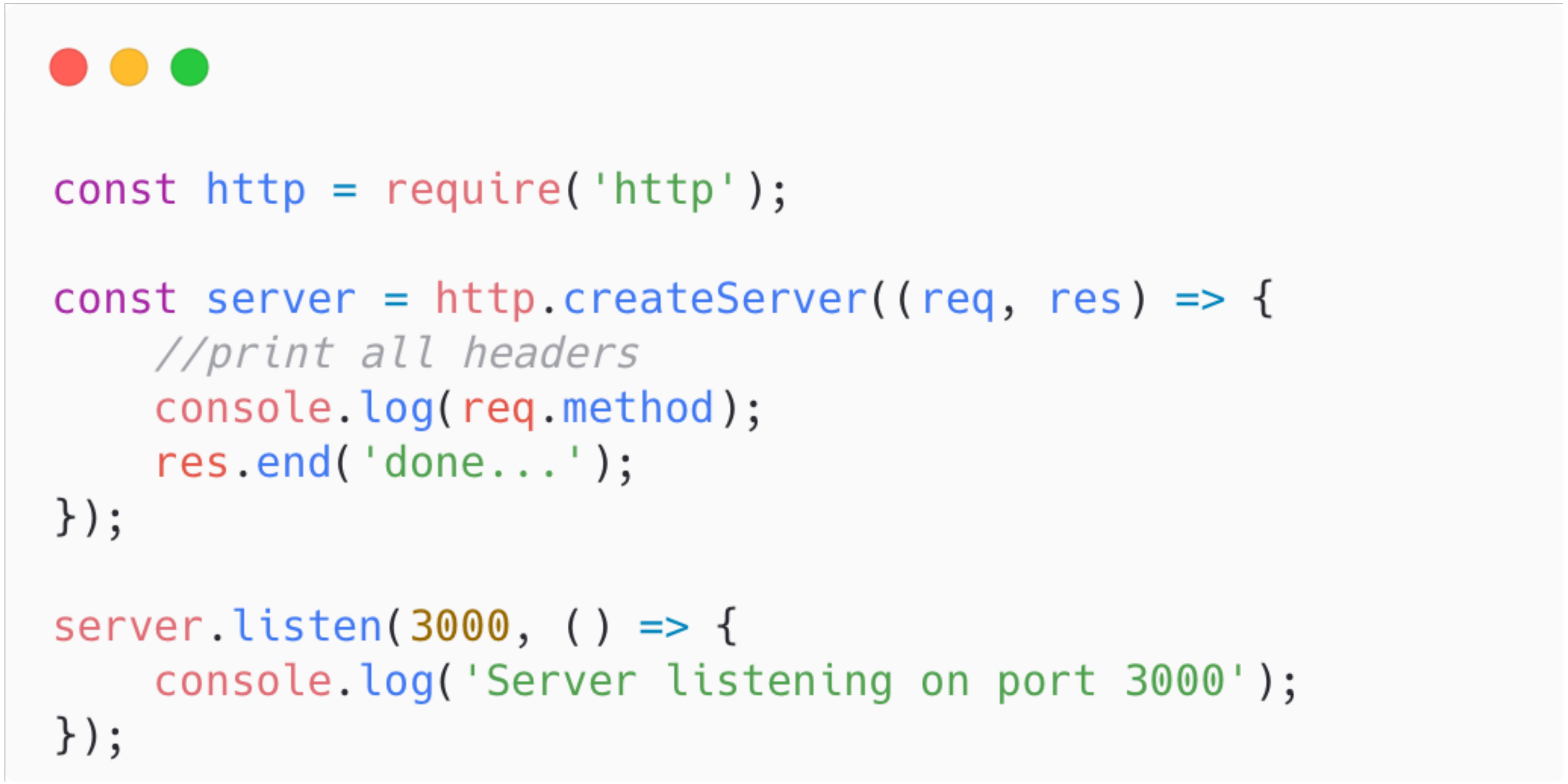
```
const http = require('http');

const server = http.createServer((req, res) => {
    res.writeHead(200, {'Content-Type': 'text/plain'});
    const path = req.url;
    if (path === '/') {
        res.write('Home page');
    } else if (path === '/api') {
        res.write('API');
    } else {
        res.writeHead(404);
        res.end(`Page not found: ${path}`);
    }
    res.end();
});

server.listen(3000, () => {
    console.log('Server listening on port 3000');
});
```

Handle HTTP requests

req.method: *the HTTP method used in the request (e.g. GET, POST, etc.)*



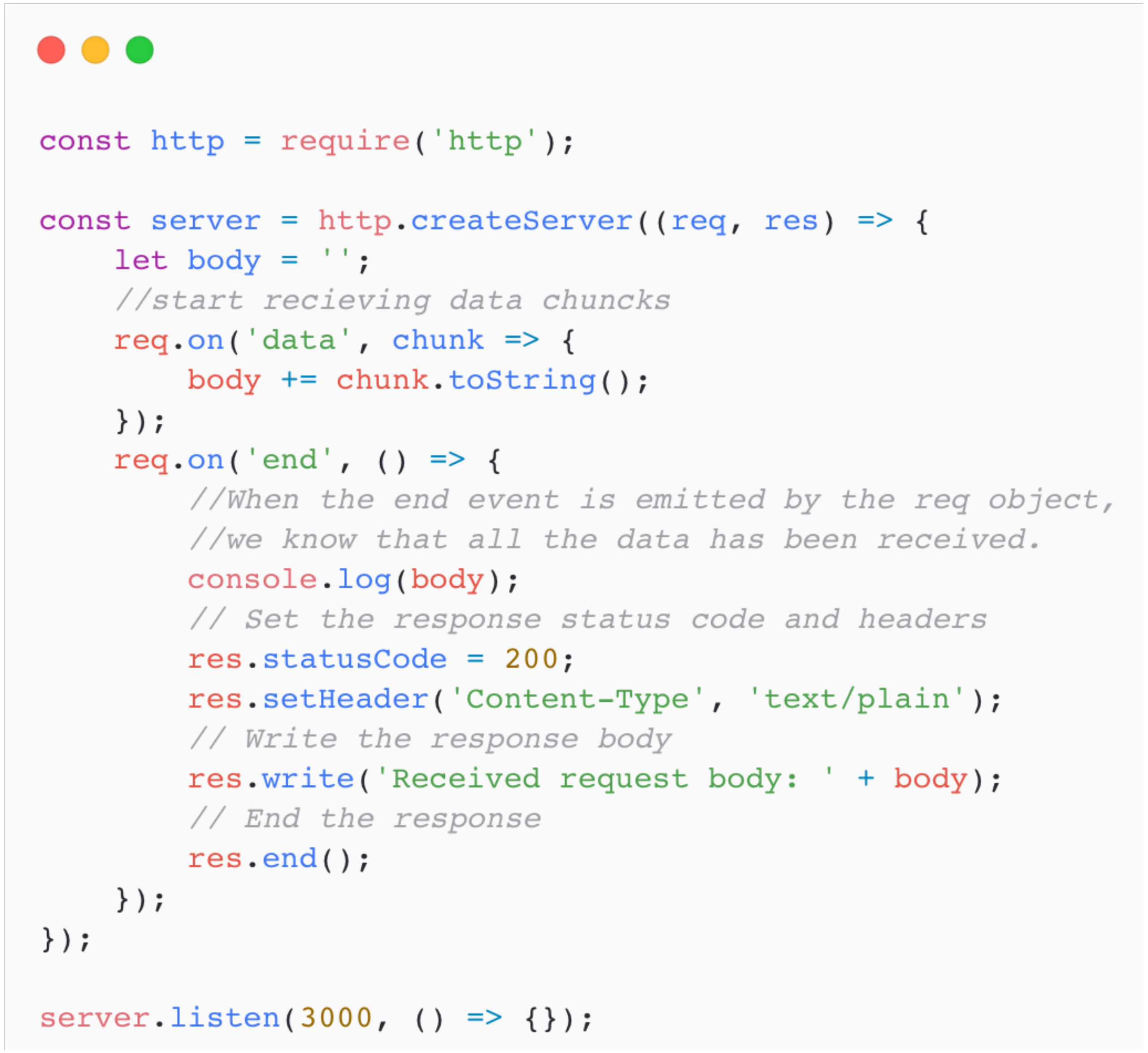
```
const http = require('http');

const server = http.createServer((req, res) => {
    //print all headers
    console.log(req.method);
    res.end('done...');
});

server.listen(3000, () => {
    console.log('Server listening on port 3000');
});
```

Handle HTTP requests

req.body: *the request body*



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

const server = http.createServer((req, res) => {
    let body = '';
    //start recieving data chuncks
    req.on('data', chunk => {
        body += chunk.toString();
    });
    req.on('end', () => {
        //When the end event is emitted by the req object,
        //we know that all the data has been received.
        console.log(body);
        // Set the response status code and headers
        res.statusCode = 200;
        res.setHeader('Content-Type', 'text/plain');
        // Write the response body
        res.write('Received request body: ' + body);
        // End the response
        res.end();
    });
});

server.listen(3000, () => {});
```