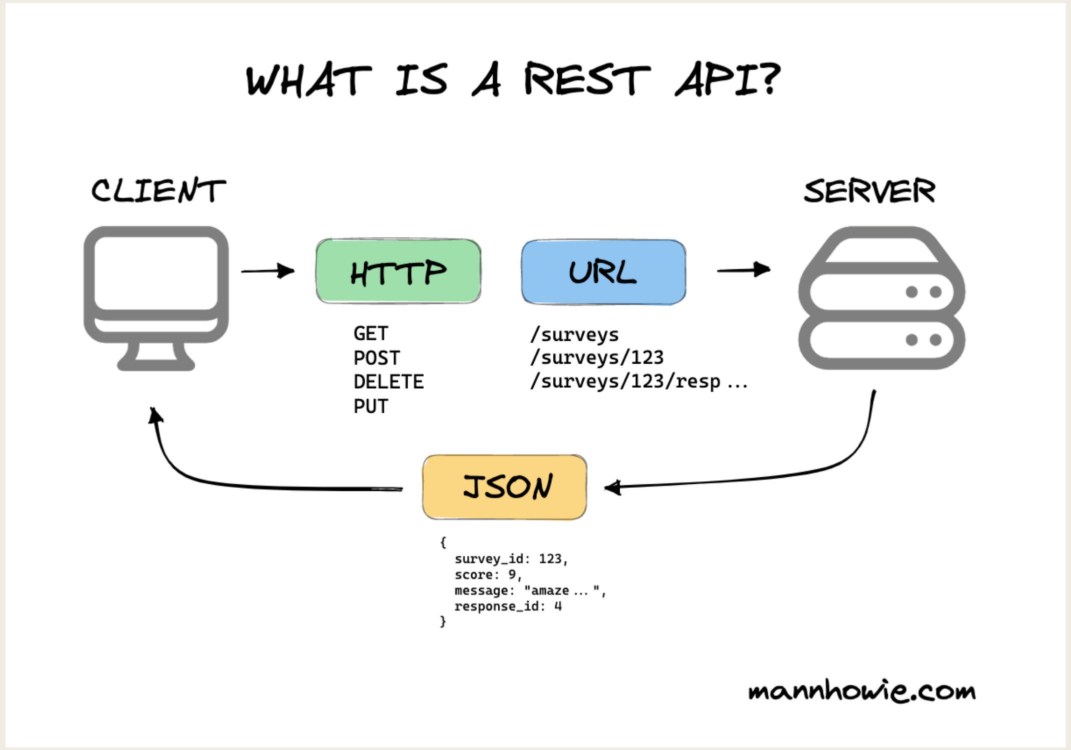
RESTful APIs

Lab 9 – 26 Mar 2025

But then why REST?

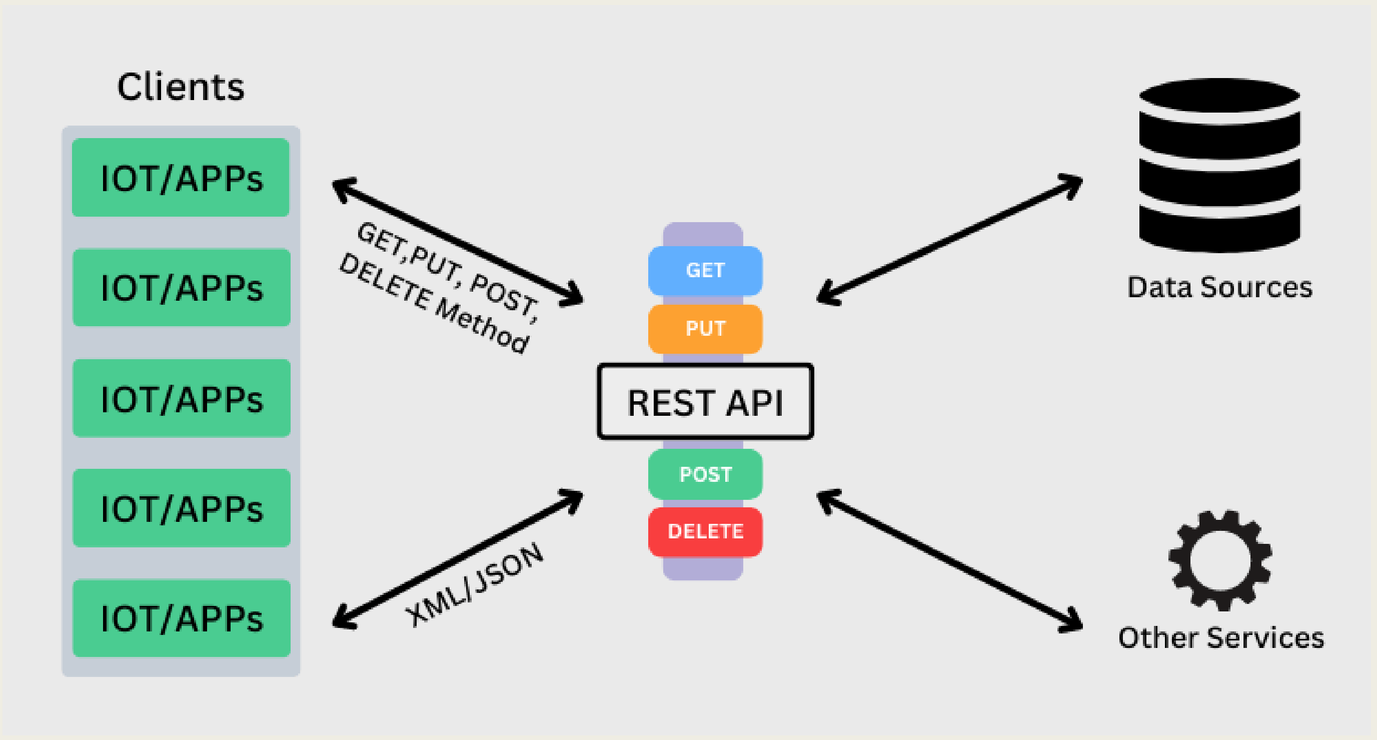
- REST - Representational State Transfer
- Widely used API development framework
- Built using the REST architecture
- Request – response model
- Supports different HTTP methods like get, post, delete, put and patch
- Additional information can be passed in a *body/payload*



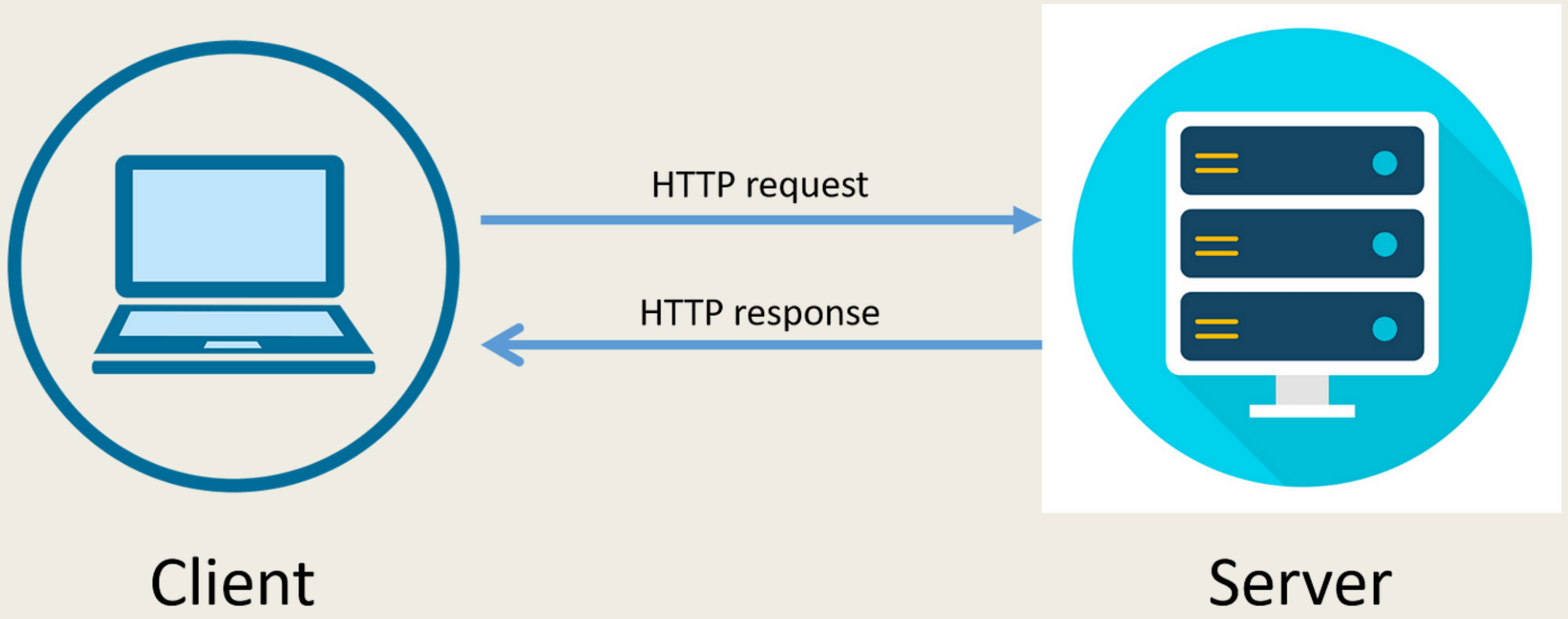
```
{
  survey_id: 123,
  score: 9,
  message: "amaze ... ",
  response_id: 4
}
```

REST Architecture

- Uniform interface
- Statelessness
  - How do you store the chat history?
- Client-server based
- Layered system
- Cacheability
- Code on demand



Request-response model



Request-response model

Every request has these components

- URI / Endpoint
- Method
- Headers
- Parameters
- Payload

A response typically has

- Status code
- Headers
- Body

HTTPS Request Example

https://www.iiit.ac.in/students --- Endpoint

GET, POST, PUT, DELETE --- Method

Content-Type: "application/json" --- Header

Create a new student --- Body
with name "Amey Karan"

HTTPS Response Example

HTTPS 1.1 200 OK --- Status

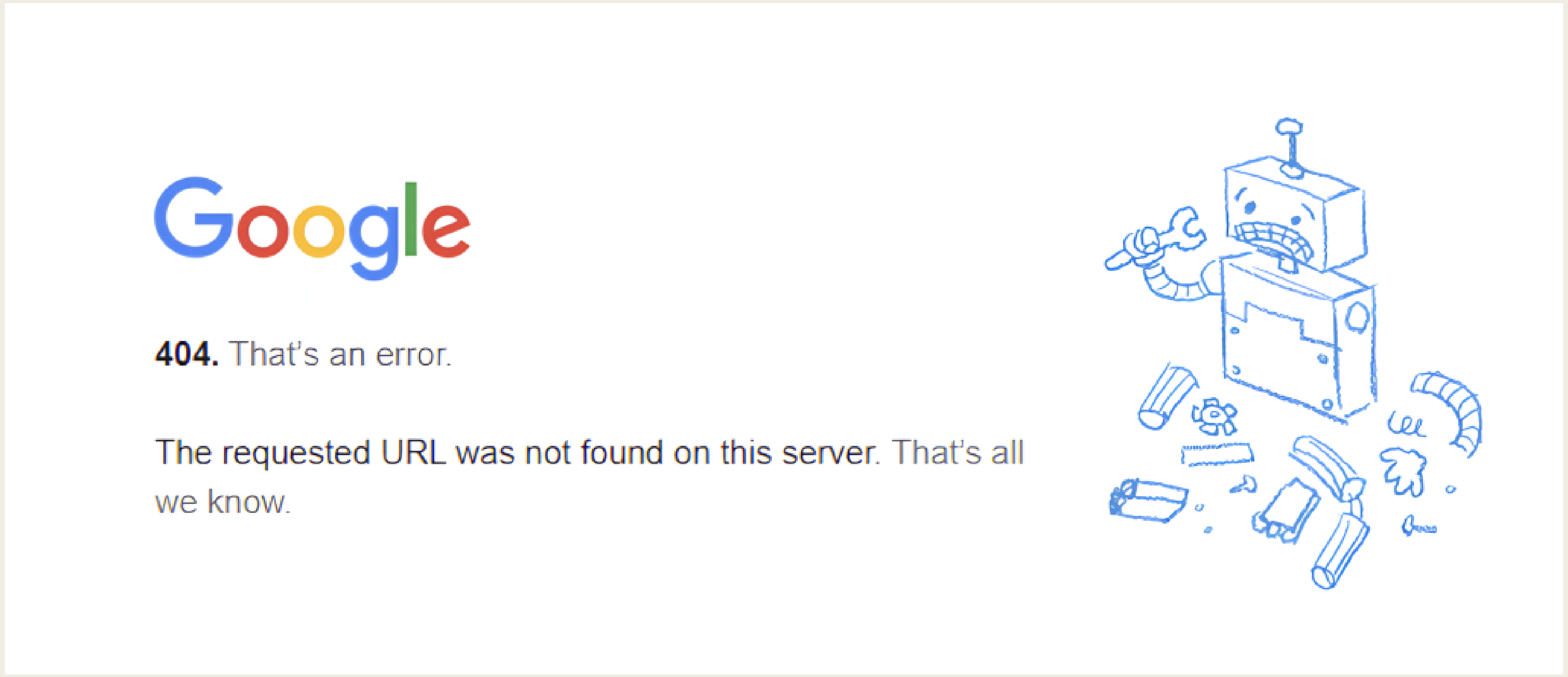
Content-Type: text/html --- Header
Charset=utf-8

<h1> Hello Amey👋 </h1> --- Body

HTTP Methods

Method	CRUD	Entire Collection (e.g. /customers)	Specific Item (e.g. /customers/{id})
POST	Create	201 (Created), 'Location' header with link to /customers/{id} containing new ID.	404 (Not Found), 409 (Conflict) if resource already exists.
GET	Read/Retrieve	200 (OK), list of customers. Use pagination, sorting and filtering to navigate big lists.	200 (OK), single customer. 404 (Not Found), if ID not found or invalid.
PUT	Update/Replace	405 (Method Not Allowed), unless you want to update/replace every resource in the entire collection.	200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid.
PATCH	Update/Modify	405 (Method Not Allowed), unless you want to modify the collection itself. Which is possible if operating on the collection as a whole.	200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid.
DELETE	Delete	405 (Method Not Allowed), unless you want to delete the whole collection—not often desirable.	200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid.

HTTP Status Codes

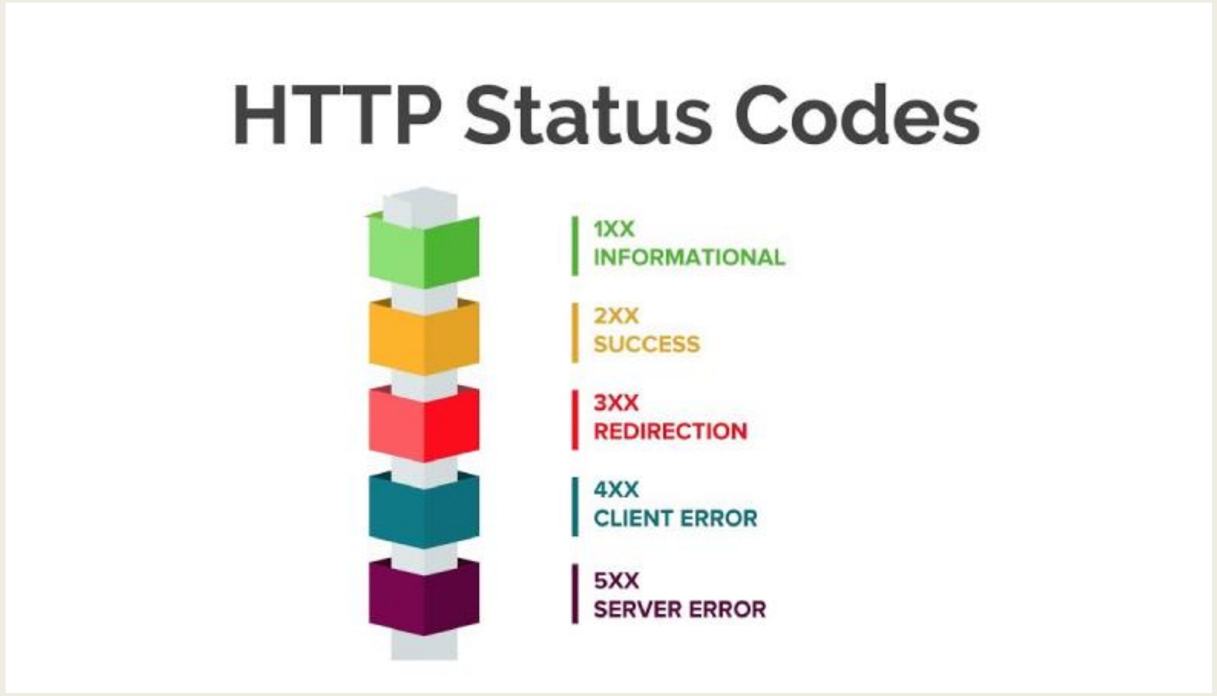


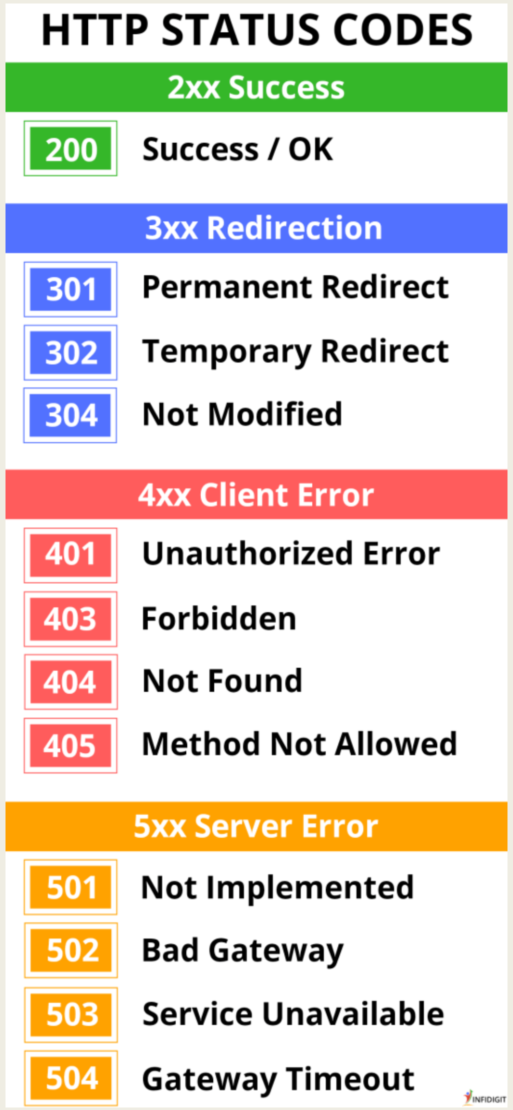
Google

404. That's an error.

The requested URL was not found on this server. That's all we know.

HTTP Status Codes





HTTP STATUS CODES

2xx Success	
200	Success / OK

3xx Redirection	
301	Permanent Redirect
302	Temporary Redirect
304	Not Modified

4xx Client Error	
401	Unauthorized Error
403	Forbidden
404	Not Found
405	Method Not Allowed

5xx Server Error	
501	Not Implemented
502	Bad Gateway
503	Service Unavailable
504	Gateway Timeout

https://blog.postman.com/what-are-http-status-codes/

REST != HTTP

	HTTP	REST
Definition	A protocol for transferring data	An architectural style for designing APIs
Focus	Communication between client and server	How APIs should be structured and interact
Relationship	REST often uses HTTP as its underlying protocol	REST is not a protocol itself, but an architectural style that can leverage HTTP

RESTful API authentication methods

Basic Authentication - client sends a username and password encoded in Base64 within the HTTP request header. While simple to implement, it is less secure unless used over HTTPS, as credentials can be easily intercepted.

Bearer Authentication (Token-Based Authentication) - involves issuing a token to the client upon successful login. The client must include this token in the request headers for subsequent requests. Tokens can be short-lived and dynamically generated, enhancing security by minimizing interception risks.

API Key Authentication - An API key is a unique identifier assigned to each client. The client includes this key in their requests, typically in the header or as a query parameter. While straightforward, API keys can be vulnerable to theft if not transmitted securely.

OAuth - more complex and secure method that allows users to grant third-party applications access to their resources without sharing passwords. It involves multiple steps, including obtaining an access token that is used for authorization. OAuth 2.0 is widely adopted for applications needing access to user data from external services like Google or Facebook.

RESTful API security features

- **Content Security Policy (CSP)**
  - Mitigates **Cross-Site Scripting (XSS)** and **data injection attacks.**
  - Allows developers to restrict dynamic resources (e.g., scripts, styles, images) to **trusted domains.**
  - Ensures only authorized content is executed by the browser.
- **Cross-Origin Resource Sharing (CORS)**
  - Enables secure access to **restricted resources** from other domains.
  - Prevents **unauthorized access** to sensitive data.
  - Requires proper server configuration:
    - **Access-Control-Allow-Origin**: Specifies permitted origins.
    - **Access-Control-Allow-Methods**: Defines allowed HTTP methods.
    - **Access-Control-Allow-Headers**: Lists headers that can be used.
    - **Same-Origin Policy**
  - Restricts cross-origin interaction of scripts and resources.
  - Prevents malicious access to **sensitive data** from another site.
  - Limits legitimate cross-origin requests unless **CORS** is properly configured.

Let's get our hands dirty

Breakdown of the Mess portal using Postman.

Activity

Hop on to

http://[REDACTED]:8000

Intro to NodeJS

What is it?

- NodeJs is a JavaScript runtime that’s built on Google Chrome’s V8 JavaScript engine.
- So, what does this mean?
- When you have written a plain JS code, you must have run it in the console of your browser, where the JS could run.
- NodeJS is a similar kind of runtime environment, which allows you to run your JS code outside the browser (May be on your terminal itself).

Quick Checks

- Ensure you have Node installed on your laptops by entering the following commands on your terminal:
- `node -v`
- `npm -v`
- If you could see the versions, you are good to go, else, you need to install node.
- Since, we have asked you to install node before coming here, we assume everyone has Node installed on their laptops and are good to go.

console.log()

- Create a file named “basics.js”.
- We will use this file to run all our basic JS commands until this presentation ends.
- Open the file.
- Type the following:
  - console.log(“Welcome to NodeJS”)
- Now run this using the command “node basics.js”
- What is the output that you observe?

Modules

- NodeJS uses a modular architecture.
- There are 2 types of modules:
  - Built-in: These come default with Node, and you are not required to install them separately.
  - User-defined: These are created by developers and are available to install.
- We use “require” to import the modules we need.
  - For example, below syntax is used to import a module named “path”.
  - const path = require('path');

Global Objects

- A **global object** in Node.js is an object that's always available in every module without needing to import it. Examples include `__dirname` (current module's directory), `__filename` (full path of the module), and `process` (information about the Node.js process). These are like built-in helpers to make coding easier.
- Just to understand, try these:
- console.log('Directory:', __dirname);
- console.log('File:', __filename);
- What is the output?

Callbacks

- **A callback is a function passed as an argument to another function.**
- This mechanism allows JavaScript to perform tasks like reading files, making HTTP requests, or waiting for user input without blocking the execution of the program.
- Callback functions help manage asynchronous operations, ensuring that the code continues to run smoothly without waiting for tasks to complete.

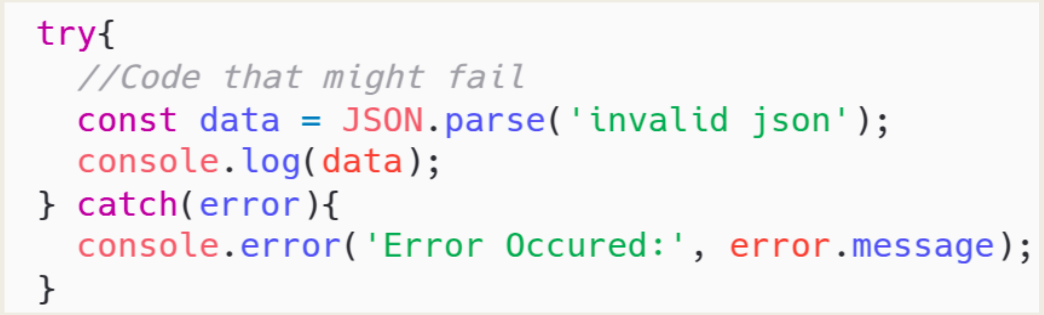
```
function greet(name, callback) {
  console.log(`Hello, ${name}!`);
  callback();
}
function sayGoodbye() {
  console.log("Goodbye!");
}
greet("Alice", sayGoodbye);
```

Error Handling

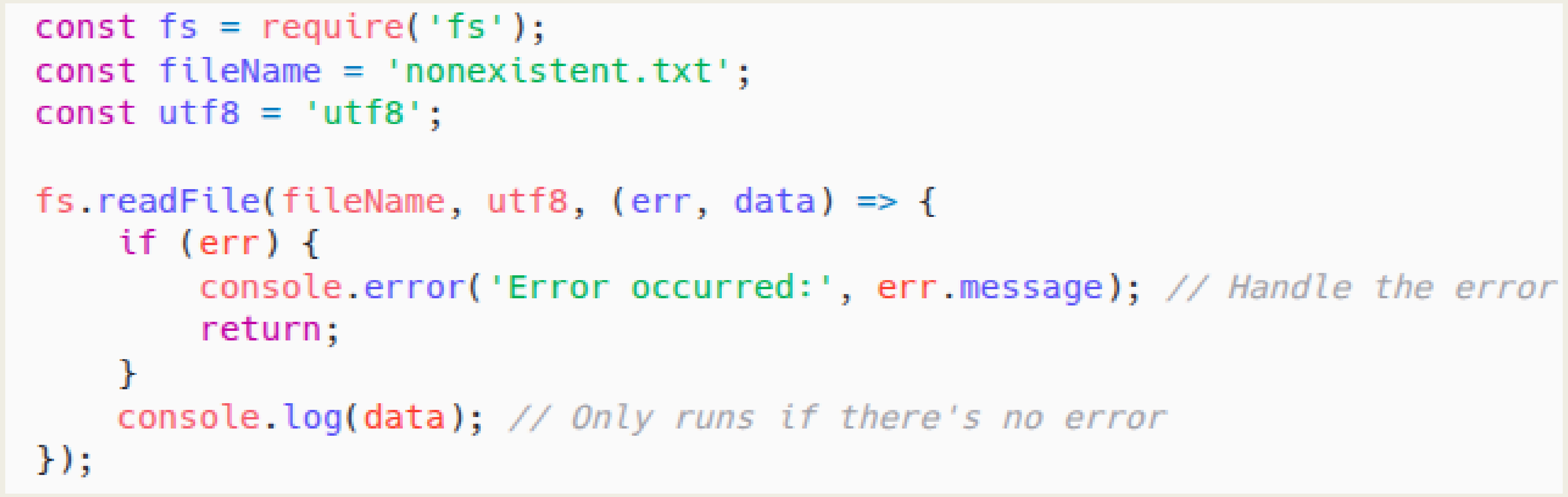
- It’s like wrapping your code in a safety net—so if something breaks (e.g., a file is missing), you can "catch" the error and decide what to do next instead of the program just stopping.
- But how can we do that?
- We’ll do it using callbacks or try…catch. Pass an err argument to a callback function. Check if err exists to handle the error.
- But what are callbacks?
- A callback is a function that you pass to another function to be executed later, usually after some task is done. It's like saying, "Once you're done, call me back!"

Error Handling

- What is try…catch
- It’s like a protective wrapper around your code. If an error happens inside the try block, the catch block will handle it.



```
try{
  //Code that might fail
  const data = JSON.parse('invalid json');
  console.log(data);
} catch(error){
  console.error('Error Occured:', error.message);
}
```



```
const fs = require('fs');
const fileName = 'nonexistent.txt';
const utf8 = 'utf8';

fs.readFile(fileName, utf8, (err, data) => {
    if (err) {
        console.error('Error occurred:', err.message); // Handle the error
        return;
    }
    console.log(data); // Only runs if there's no error
});
```

File System Module

- But, wait! Did you observe something called fs in the previous example?
- What is it?
- It is a module that is used to work with file system. (Simply like file handling in C that you have learnt).
- Import this library by using require(‘fs’)
- Some of the common methods in this module are:
  - fs.readFile()
  - fs.writeFile()
- Let us look at some examples.

HTTP Module

- The http module is used to create servers and handle HTTP requests and responses. It's the core module for building web servers in Node.js.
- Various functions of HTTP module are:
  - Creating a server
  - Handling Routes
  - Handling JSON data
  - Making HTTP requests
- Let us look at basic example on creating server.
- Explore the modules in NodeJS at their official documentation on Modules in NodeJS. Link can be found here.

What the hell is asynchronous ???

- **Synchronous (Sync):**
- Imagine you go to a ice-cream store and order an ice cream. After you place the order, you are not allowed to do anything except waiting for the ice cream to be delivered.
- **Asynchronous (Async):**
- Now Imagine you go to a ice-cream store and order an ice cream. While the ice cream is being made, you go and do grocery shopping and give your laundry, and then when the store makes your order, you go and collect it.
- The 2nd scenario is where you do not wait for a task to be completed and do your work but rather do it while waiting for the 1st task to complete.
  2nd Scenario is asynchronous.

In simple language

- **Synchronous**- Waiting for the task to complete, before executing the next steps
- Executing one task at one time
- **Asynchronous**- Does not wait for previous task to complete, before executing the next steps
- Allow other tasks to be executed while waiting for the first task to finish.

Callbacks for the same

- But, As the complexity of the code increases,using multiple nested callbacks can become complex and hard to manage.

```
function greet(name, callback) {
  console.log(`Hello, ${name}!`);
  callback();
}
function sayGoodbye() {
  console.log("Goodbye!");
}
greet("Alice", sayGoodbye);
```

Some Syntax

```
// ES5
var x = function(x, y) {
   return x * y;
}


// ES6
const x = (x, y) => x * y;
```

Both do the same thing. JS ES6(2015) is a javascript version that provides the arrow function.That is just a syntax that is used to write code faster.

Promises?

- "**Producing code**" is code that can take some time
- "**Consuming code**" is code that must wait for the result
- **A Promise is an Object that links Producing code and Consuming code**
- **Promises help in writing Async code. (This is an old tool now , are is used under the hood)**

A JavaScript Promise object can be:

- Pending
- Fulfilled
- Rejected

Async Function

"async and await make promises easier to write"

async makes a function return a Promise

await makes a function wait for a Promise

```
async function example()
{
        return “hello”
}
```

```
function fetchData() {
    return new Promise((resolve) => {
        setTimeout(() => resolve("Data
fetched"), 1000);
    });
}

async function getData() {
    console.log("Fetching...");
    const data = await fetchData();
    console.log(data);
}

getData();
```