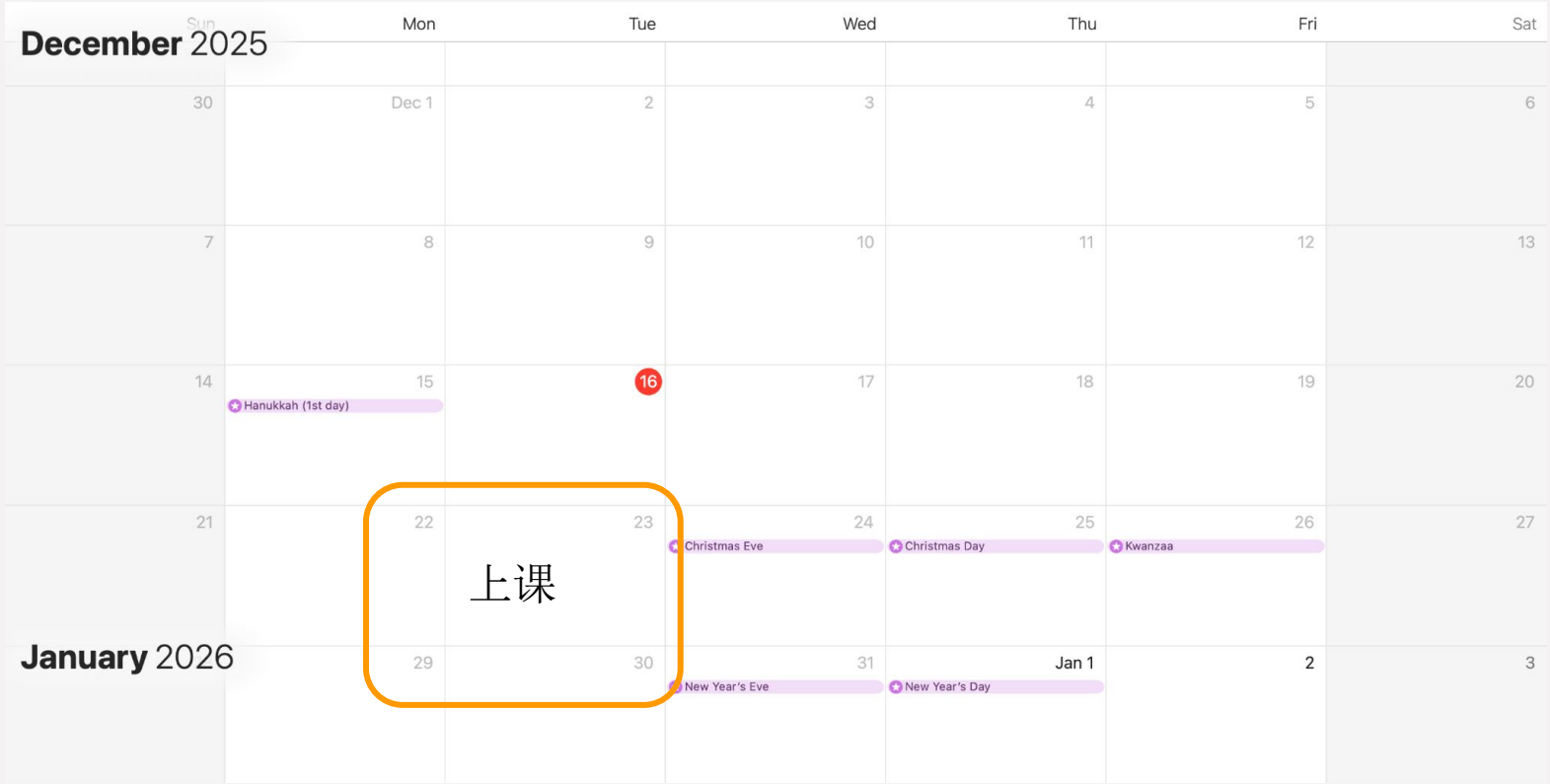
05. Javascript (Part3)

Group 1	Tingwei	Liu
	Haoyu	Li
	Shaolong	Li
Group 2	Sijun	Hua
	Weihang	Guo
	Chieh Jui	Lee
	Weiren	Feng
Group 3	Chunjingwen	Cui
	Huimin	Mu
	Rongwei	Ji
Group 4	Jiahao	Liang
	Lingyu	Xu
	Di	Wang
Group 5	Ziyue	Fan
	Fangqin	Li
	Ruiqi	Wang
	Ruohan	Wang
Group 6	Haozhong	Xue
	Kanghong	Zhao
	Chia Hsiang jia xiang	Wu
	jingwei	Ma

Schedule


December 2025
Sun
Mon
Tue
Wed
Thu
Fri
Sat
30
Dec 1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
Hanukkah (1st day)
21
22
23
24
25
26
27
Christmas Eve
Christmas Day
Kwanzaa
上课
January 2026
29
30
31
Jan 1
2
3
New Year's Eve
New Year's Day

Schedule

- Dec 15 - Dec 19: Javascript + React
- Dec 22 - Dec 26: React + Redux => interview coding questions mock
- Dec 29 - Jan 2: Node + Test => Full Stack Website Clone
- Jan 5 - Jan 9: Other Topics + Resume + Final Mock

Outline

- OOP
- This keyword
- Call, Apply, Bind
- JavaScript Fetch API
- Event Loop

Attention!

This training is **interview-driven**.

To become a qualified developer, *ChatGPT* and *YouTube* will always be your best friends.

Outline

- OOP
- This keyword
- Call, Apply, Bind
- JavaScript Fetch API
- Event Loop

OOP

- **Object-oriented programming (OOP)**: a programming paradigm based on the concept of classes, objects, and four main features

Principle	Description	Example
Encapsulation	Bundling data and methods that operate on the data	`class` with private variables
Abstraction	Hiding complex implementation details	Only exposing essential methods
Inheritance	A class can inherit properties and methods from another	`extends` keyword
Polymorphism	Ability to use a single interface for different types	Method overriding

This keyword

- this refers to the **context in which a function is called**
  - this is decided by how the function is called, **NOT** where a function is defined
- Keyword function: dynamic, depends on how the function is called
  - changes between strict and non-strict modes
    - this refers to window vs undefined
- Object methods
  - this refers to the object itself when called with dot notation
- **Arrow function**: Arrow functions do **NOT** have their own this
  - They inherit this from where they are defined(outer scope)
  - Arrow function inside an object: NOT bind to obj

Call, Apply, Bind

- Used to control what this refers to when a function runs
- Only work on regular functions (**NOT** arrow functions)
  - **call()**: invoke immediately (args one by one)
    - func.call(thisArg, arg1, arg2, ...)
  - **apply():** invoke immediately (args as array)
    - func.apply(thisArg, [argsArray])
  - **bind()**: returns a new function (does **NOT** execute)
    - func.bind(thisArg, arg1, arg2, ...)
    - useful for event handlers, callbacks

Call, Apply, Bind

Method	Purpose	Arguments	`this` Effect	Return Value
`call`	Invoke function immediately with custom `this`	`thisArg, arg1, arg2, ...`	Sets `this` during call	Function's return value
`apply`	Same as `call`, but arguments are in an array	`thisArg, [argsArray]`	Sets `this` during call	Function's return value
`bind`	Returns a new function with bound `this`	`thisArg, arg1, arg2, ...`	Sets `this` permanently	**New function**

Synchronous vs Asynchronous

- JavaScript is a **single-threaded** programming language
  - JavaScript can do only one thing at a single point in time
- **Synchronous**: executed line-by-line in the order that it is written, and one line of code must finish execution before the next line is executed
  - Long-running operations will block code execution
    - e.g. loading a page that requests data from an API
- **Asynchronous**: a line of code doesn't need to finish execution before the next line is executed
  - Prevents long-running operations from blocking execution
  - Defers execution via **Web APIs**

Web API

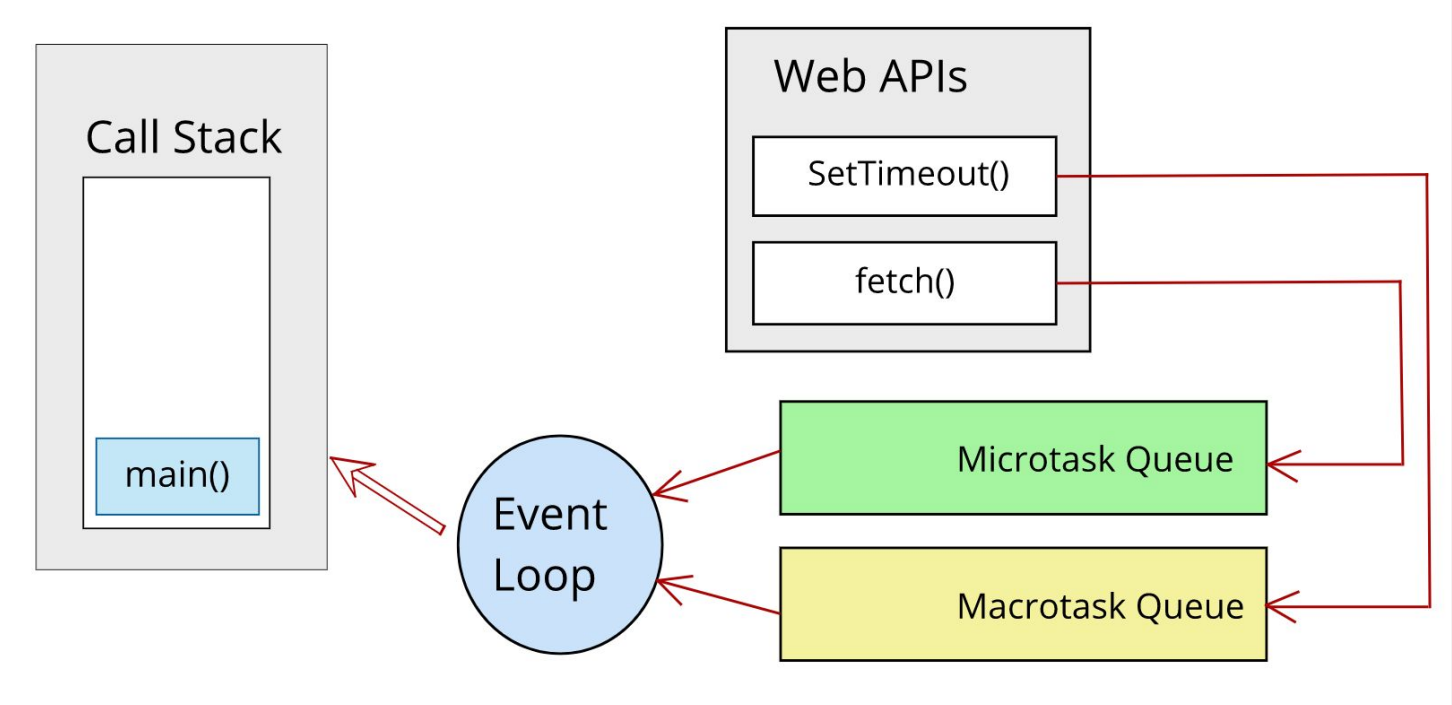
- Web APIs are features provided by the browser (or Node.js), not by the JavaScript language itself
    - Extend JavaScript’s power and handle tasks **asynchronously**

Category	Web API Feature	Description
Timers	`setTimeout`, `setInterval`	Delay or repeat code execution
Networking	`fetch`, `XMLHttpRequest`	Make HTTP requests
DOM	`document`, `querySelector`	Manipulate HTML elements
Events	`addEventListener`	Listen to user actions
Storage	`localStorage`, `sessionStorage`	Store data in the browser
Media	`Audio`, `Video`, `MediaDevices`	Play or capture media
Location	`navigator.geolocation`	Get user's physical location

Event Loop

- The Event Loop is the mechanism that lets JavaScript handle asynchronous operations (like timers, network requests, and user events) even though JavaScript itself runs in a single thread
  - Call Stack: LIFO
  - Callback/Task/Message Queue
    - Macrotask Queue (setTimeout, setInterval, I/O) - **Lower-priority queue**
    - Microtask Queue (**promises**) - **Higher-priority queue**
  - (Repeatedly) On empty Call Stack:
    - Is the call stack empty? Are there any tasks in the callback queue?
    - If both true, it moves the task from queue to stack and executes it
  - Order: synchronous > **promises** > other callbacks

Event Loop

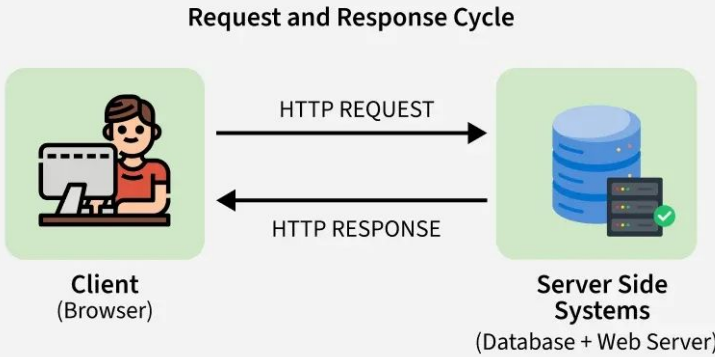

Call Stack
main()
Web APIs
SetTimeout()
fetch()
Event Loop
Microtask Queue
Macrotask Queue

HTTP Request Methods

- GET: Fetch Data
  - No request body
- POST: Create New Resource
  - Sends data in request body
- PUT: Update/Replace Entire Resource
  - Replaces the entire object on the server
  - Sends full object in the body
  - **Idempotent**: sending the same PUT request multiple times has the same result
- DELETE: Remove Data
  - Deletes a resource by ID or URL, Often doesn't need a body

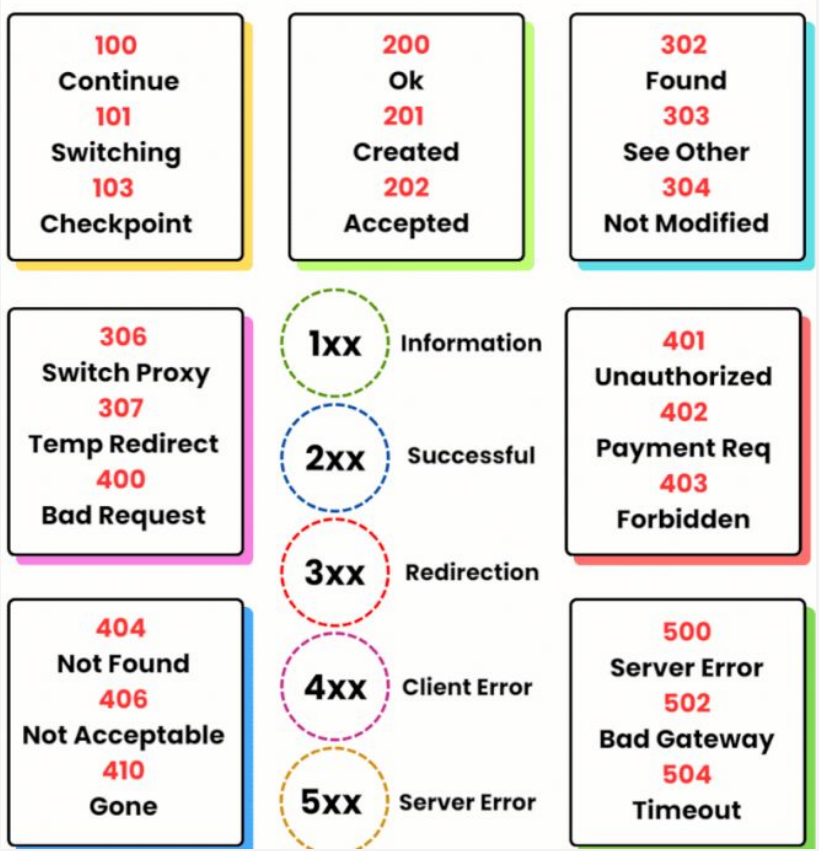
request-response cycle

- The Request-Response Cycle is the complete process that happens when a client (like your browser) sends a request to a server and gets a response back
  - Client sends an HTTP request
    - fetch, axios, etc
  - Server receives request and processes it
  - Server sends HTTP response
    - Status + Json data
    - Status Codes
  - JS handles response, updates UI or state



HTTP Status Code

- HTTP status codes are standard responses sent by a server to tell the client what happened with a request.
- They are grouped into 5 classes



Callback Function

- Callback Function: A function that is passed to another function and called later
- **callback hell**: Too many nested callbacks

```
a(() => {
  b(() => {
    c(() => {
      d();
    });
  });
});
```

Fetch

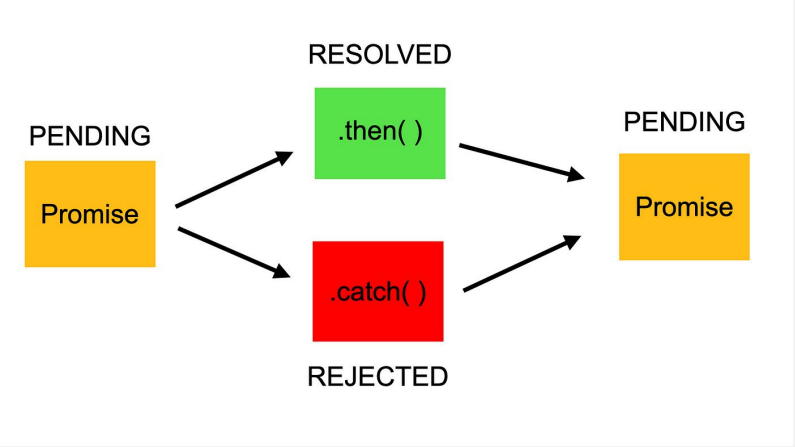
- A **built-in** JavaScript function that allows you to make asynchronous HTTP requests to servers
  - It returns a **Promise**, making it easy to work with using **.then()** or **async/await**

```
fetch(url, {
  method: "POST", // default GET
  headers: {...},
  body: {...},
})
.then(response => response.json())
.then(data => { console.log(data); ...// handle data})
.catch(error => {
  // handle the error
})
.finally(()=>{}); // optional
```

```
try {
  const res = await fetch("/api/data");
  const data = await res.json();
  // handle data
  console.log(data);
} catch (err) {
  console.error(err);
  // handle the error
}
```

Promises

- A Promise is an object that represents the eventual completion or failure of an asynchronous operation and its resulting value
- 3 states:
  - **Pending**: initial state, neither fulfilled nor rejected
  - **Fulfilled**: meaning that the operation was completed successfully **resolve()**
  - **Rejected**: meaning that the operation failed. **reject()**




```
let completed = true;
let learnJS = new Promise((resolve, reject) => {
  if (completed) {
    resolve("I have completed learning JS.");
  } else {
    reject("I haven't completed learning JS yet.");
  }
});
```

Promises

- .then(): what to do when it succeeds
  - Chaining then
- .catch(): Error Handling
- .finally(): optional, always runs
- Promise.all()
  - succeeds only if ALL Promises succeed
  - If one fails, whole thing fails
- Promise.allSettled()
  - waits for all Promises, no matter success/failure

```
fetchUser()       → Promise(pending)
    ↓ resolve
.then(...)        → Promise(pending)
    ↓ resolve
.then(...)        → Promise(pending)
    ↓ resolve
.catch(...)       → if any above fails (rejected)
    ↓ (optional)
.finally(...)     → always runs last
```

async and await

- async and await are syntax built on top of Promises to make asynchronous code easier to read and write
    - Look like normal synchronous code
- async
    - Declares a function that always returns a Promise
- await
    - Pauses the function until a Promise settles (resolves or rejects)
        - await only works inside async functions
- Async and wait should be used in together
- Error Handling: try...catch

Handle asynchronous operations

- **Callback function**
  - **Callback Hell (ES5)**: nested callback functions whose arguments are results of the outer callback
    - Difficult to read, maintain, debug
- **Promises (ES6):**
  - **Promise chain:** consecutively invoking .**then()** on a promise
    - Avoid callback hell
    - Cleaner, more readable, easier debugging & error handling
- **Async/Await:**
  - behave like **synchronous code**
  - Avoid callback hell
  - Avoid .then()/.catch() chaining
  - Improve readability and error handling

Handle asynchronous operations

Feature	Callback	Promise	async/await
Introduced In	Original JS	ES6 (2015)	ES8 (2017)
Syntax Style	Nested	Chained `.then()`	Synchronous-like (`await`)
Error Handling	Manual, inside each callback	`.catch()` chain	`try...catch` block
Readability	❌ Poor (callback hell)	✅ Better than callbacks	✅ Best and cleanest
Debugging	❌ Hard	🟡 Medium	✅ Easy (like sync code)
Return Value	N/A	Promise	Promise
Use With `await`	❌ No	✅ Yes	✅ Yes

Error Handling

- Writing code that processes errors to prevent the program from terminating
- try...catch: For synchronous code or async/await
  - catch block: runs only if an error occurs
- .catch(): For Promises
- throw
  - implicitly, by the program (reference error, variable not declared)
  - explicitly, by code:
    - throw new Error("xxx")
    - Promise.reject("xxx")

Additional materials

- https://jsonplaceholder.typicode.com/
- AJAX, XMLHttpRequest (XHR), Fetch vs XHR, Fetch vs Axios

Homework - Requirement

- 全程recording: 八股闭眼 & 摘耳机, 无AI辅助
- 全程recording: Coding可适当AI辅助, 但需写完(边解释边写), 模拟真实面试情景
- 录音提交地址: https://drive.google.com/drive/folders/1Mrm341uOIS8c2Ty6NwYvvSybHa6hESem?usp=drive_link
- 提交格式: 小组+日期, 例如Group1_12/07/2025
- 提交截止时间: due第二天5PM (周四作业递延到下周一)

Homework - Class Code

- Write the code as taught in class, take a screenshot of your code, and post it in the Wechat group. **It's due the same night.**

Homework - Short Answer Questions

1. What is Object-Oriented Programming (OOP)?
2. Explain the this keyword in JavaScript.
3. What are the differences between call, apply & bind?
4. What are the disadvantages of synchronous code?
5. What is asynchronous code in JavaScript? How does JavaScript achieve asynchronous code?
6. What is the callback queue?
7. What does the event loop do? What data structures does it use?
8. What is an HTTP request and HTTP response?
9. How many HTTP methods are there? Explain each one.
10. What is the difference between GET and POST? What about POST and PUT?

Homework - Short Answer Questions

11. Could you explain the different classes of HTTP status codes? What are some common status codes?
12. What is a Promise?
    - What is promise chaining
13. Explain the three states of a Promise.
14. What is callback hell?
15. What is the advantage of Promises over callbacks?
16. Explain Promise.all() vs Promise.allSettled().
17. What is the Microtask Queue?
18. What is async & await? How do we use them?
19. Explain Microtasks and Macrotasks in the Event Loop.

Homework - Short Answer Questions

20. What is the advantage of Promises over callbacks?
21. What is the purpose of the finally() method in a Promise chain?
22. What are the main rules of promise
23. How to handle asynchronous operations

Homework - Coding Questions

1. Use HTML/CSS/JS to solve the following problems. Please follow best practices when you write the code so that it would be easily readable, maintainable, and efficient.
    - [Part 1] Given a url “https://jsonplaceholder.typicode.com/users”, send a GET request to display the data on the page in a “table”. Errors should be handled properly.

Homework - Coding Questions

- [Part 2] Create a text input box and a search button. When you input a user ID and click search, it should display that user's information, posts, and todos all in the same page in a "list" with the format of key: value.
  - Hint: Promise.all() or Promise.allSettled()
  - For example, when the user types 2, display the data from the following urls:
    - https://jsonplaceholder.typicode.com/users/2
    - https://jsonplaceholder.typicode.com/posts?userId=2
    - https://jsonplaceholder.typicode.com/todos?userId=2
    - If the user ID is invalid (no data in the response), there should be an error message says `User was not found. Please try another user ID` and then clear the input box.

Homework - Coding Questions

Put the three parts on a single page, similar to the example below: