03. Javascript (Part1)

Attention!

This training is **interview-driven**.

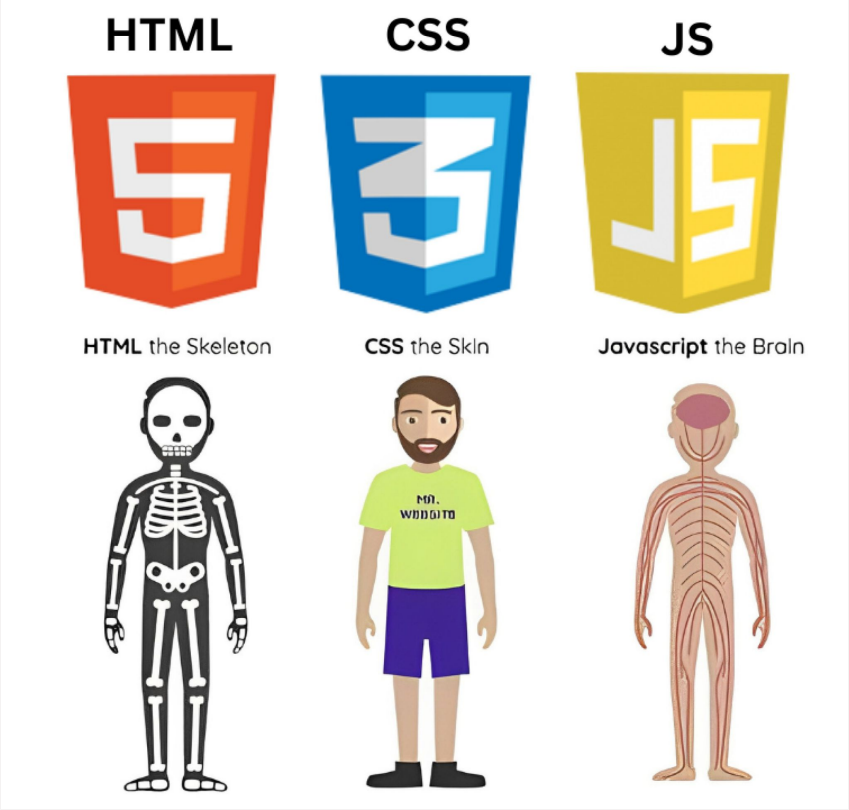
To become a qualified developer, *ChatGPT* and *YouTube* will always be your best friends.

Outline

- Intro
- Scope
- Hoisting
- Variable
- Operators
- Short-circuiting
- Data Type
- Template Literal
- Object
- Array
- Spread Operator
- Map
- Set

Javascript

- JavaScript is a high-level, interpreted programming language
  - HTML = structure (skeleton)
  - CSS = style (clothing)
  - JavaScript = behavior (**movement**)
- JS runs in the browser's JavaScript engine (e.g., **V8 in Chrome**)
- Modern JS uses **ES6+** features (2015 and beyond)



ECMAScript vs Javascript

- ECMAScript is the rules
- JavaScript is the actual language following those rules

ECMAScript	JavaScript
A **standard/specification**	A **programming language**
Defines how the language should work	A **real implementation** of ECMAScript
Created by ECMA International	Created by Netscape (Brendan Eich)
Versioned as ES5, ES6 (ES2015), ES7...	Includes browser APIs like `document`, `window`

How to Run Javascript

- Browser (Console)
- In a Browser HTML File: <script>
- terminal
- Online Editor
- **REPL**
  - An **interactive** environment where you can type JavaScript code, run it line-by-line, and see results instantly
    - Read → Eval → Print → Loop
    - terminal, Browser
    - Great for testing small code snippets or debugging

Scope

- Scope is the “area” of your code where a variable can be accessed
  - It decides where a variable is visible and where it is not
- Types
  - Global Scope: accessible everywhere
  - Function Scope: only visible inside that function
  - Block Scope (introduced with let/const): only visible in that block

```
let a = 10; // global

function demo() {
  console.log(a); // 10 — can access
}
```

```
function test() {
  var x = 5;
}
console.log(x); // ❌ Error — x is inside the function
```

```
if (true) {
  let y = 20;
}
console.log(y); // ❌ Error — y is block-scoped
```

Hoisting

- JavaScript moves **declarations** (but **not initializations**) to the top of their scope, **before** any code runs
- You can write code that uses a function or variable before actually declaring it
  - **keyword** function: only applies to the keyword function, function expression & arrow function are assigned to variables, will not be hoisted
  - Variable:
    - var: undefined (not error!)
    - let and const: ReferenceError

Variable

- A variable is a name that refers to a value
- Create variables with var, let, or const
  - var : function-scoped
    - Variable declared anywhere else besides a function using var will always exist in the global scope
    - Variable declared with var CAN be redeclared and reassigned
  - let : block-scoped
    - Variables declared with let CAN be re-assigned but **CANNOT** be redeclared
  - const: block-scoped
    - Variable declared with “const” **CANNOT** be redeclared or reassigned

Variable - Scope

- Always use **const** if the value should not be changed
- Only use **let** if you can't use const
- Only use **var** if you **MUST** support old browsers

	var	**let**	**const**
Scope	Global	Block	Block
Hoisted	Yes, initialized to **undefined**	Yes, not initialized, **reference error**	Yes, not initialized, **reference error**
Can reassign?	Yes	Yes	**No**
Can redeclare?	Yes	**No**	**No**

Operators

- Math operators: +, -, *, /, %, **
- Comparison operators: >, <, >=, <=, ==, ===
- Logical operators
  - ! : negates the boolean value
  - && (AND operator) : Gives True only if **both** the operands are True
  - || (OR operator): Gives True if **either** of the operands is True
    - **Short-circuiting**
- Assignment operators: =, +=, -=, *=, /=, %=
- Conditional operators
  - **Ternary Operator** (?:): condition ? a : b
- Nullish Coalescing Operator (??)
  - Use a fallback value if null or undefined

Short-circuiting

- A behavior where logical operators (&&, ||, ??) stop evaluating as soon as the result is determined
  - OR ||: Returns the **first truthy** value, or the last one if all are falsy
    - Providing default values
  - AND &&: Returns the **first falsy** value, or the last one if all are truthy
    - Conditional execution
  - Nullish Coalescing ??: Returns the right-hand value only if the left is **null** or **undefined**
    - Safely assign default values without overwriting falsy values like 0 or ""

Data Type - Primitive Types

- Store **single** values and are **immutable**
  - **Immutability:** the structure or data cannot be changed
    - **CANNOT** change their actual value, Any "modification" creates a new value
  - Primitive types
    - Number → 42, 3.14
    - String → "Hello"
    - Boolean → true / false
    - Undefined → variable declared but not assigned
    - Null → explicitly no value
    - BigInt → 123n (large integers)
    - Symbol → unique identifiers
  - **typeof**: a unary operator that takes 1 operand and evaluates to a string that represents its data type

```
let a = "abc";
let b = a;
b = "xyz";
```

a → "abc"
b → "xyz"

Undefined vs Null

- Use undefined to check if a variable is uninitialized.
- Use null when you intentionally want to clear a value or indicate “empty”.

Feature	`undefined`	`null`
Type	`"undefined"`	`"object"` (bug)
Assigned by JS	✅ automatically for uninitialized vars	❌ must be assigned manually
Meaning	Variable exists but **no value yet**	**Intentional absence of value**
Usage	Default, missing value	Explicitly empty, cleared value
Equality	`undefined == null → true`	`undefined === null → false`

Template Literal

- literals delimited with **backtick** (``) characters, allowing for multi-line strings, string interpolation with embedded expressions
  - Introduced in **ES6**
  - use backticks (`) instead of quotes
  - Basic Syntax: `` `xxxx ${xxxx}` ``
    - Multiline Strings
    - Expression Evaluation
    - Nesting Templates
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Template_literals

String Methods

Template Literal: Allow for string interpolation with embedded expressions

- .toUpperCase()
- .toLowerCase()

Method	Purpose	Example
`trim()`	Removes whitespace from both ends of the string	`" Hello ".trim()` → `"Hello"`
`indexOf(substring, fromIndex)`	Returns the first index of substring, -1 if not found	`"Hello World".indexOf("o")` → `4`
`split(separator, limit)`	Splits string into an array by separator; optional limit on number of splits	`"Hello World JS".split(" ")` → `["Hello", "World", "JS"]`
`join(separator)`	Joins array elements into a string using separator	`["Hello", "World"].join(" ")` → `"Hello World"`
`includes(searchString, position)`	Returns true if string contains the substring	`"Hello World".includes("World")` → `true`
`slice(startIndex, endIndex)`	Extracts a substring from startIndex to endIndex (not inclusive)	`"Hello World".slice(0,5)` → `"Hello"`
`startsWith(substring, position)`	Checks if string starts with the substring	`"Hello World".startsWith("Hello")` → `true`
`substr(startIndex, length)`	Extracts substring of given length from startIndex (deprecated)	`"Hello World".substr(6,5)` → `"World"`

Falsy values

- Values that evaluate to **false** when used in a boolean context
  - false, 0, empty string (‘’, “”,``), undefined, null, NaN

Data Type - Reference Types

- Store a reference to the memory **location** where the data is stored, rather than the actual value itself
  - Stored by **reference**, not by value
  - Comparison by reference
  - Mutability: Objects and arrays can be changed even if declared with const

```
let obj1 = { name: "Tom" };
let obj2 = obj1;
obj2.name = "Jerry";
```

```
css

obj1 ──┬──► { name: "Jerry" }
       │
obj2 ──┘
```

```
let obj1 = { name: "Alice" };
let obj2 = obj1;

Memory:

  ┌──────────┐
  │{ name: │
  │ "Alice"}◄──────── obj1
  └──────────┘

                ▲
                └──── obj2
```

Data Type - Reference Types

Type	Example	Notes
Object	`{ name: "Alice", age: 25 }`	General objects
Array	`[1, 2, 3]`	Arrays are objects
Function	`function greet(){}`	Functions are objects too
Date	`new Date()`	Built-in object
Map / Set / WeakMap / WeakSet	`new Map()`	Collection objects
RegExp	`/abc/`	Regular expression object

Primitive Types vs Reference Types

Feature	Primitive Types	Reference Types
Memory	Stack	Heap
Mutability	Immutable	Mutable
Assignment	By value	By reference
Examples	Number, String, Boolean, null, undefined, Symbol, BigInt	Object, Array, Function, Date, Map, Set, RegExp

Object

- An object is a collection of **key-value** pairs
  - Keys (also called properties) are strings or Symbols
    - Keys can be unquoted if they are valid identifiers (name, age)
    - Keys with spaces or special characters must be in quotes
  - Values can be any type: primitive, array, function, another object, etc
- Accessing Values
  - dot notation
  - brackets notation
- Built-in Methods
  - Object.keys: Returns an **array** of **keys**
  - Object.values: Returns an **array** of **values**
  - Object.entries: Returns an **array** of **[key, value] pairs**, each as an array

Shallow Copy vs Deep Copy

- **Shallow** copy
  - Only the first level is copied. Nested objects remain references
  - Modifying nested objects affects the original.
  - Methods
    - **Spread operator**: { ...obj } or [...arr]
    - Object.assign({}, obj)
- **Deep** copy:
  - Creates a completely independent copy, including all nested objects/arrays
  - Methods
    - structuredClone(obj) (modern JS, best way)
    - JSON.parse(JSON.stringify(obj))

Shallow Copy vs Deep Copy

Feature	Shallow Copy	Deep Copy
Levels copied	Only top level	All nested levels
Nested objects	Shared by reference	Fully independent
Changes to copy	May affect original if nested	No effect on original
Methods	`...`, `Object.assign`	`structuredClone`, `JSON.parse(JSON.stringify())`
Use case	Simple objects, no nested references	Complex objects with nested structures

Spread Operator

- …: “spreads” the elements of an array or object into another array, object, or function arguments
  - Used for **copying**, **merging**, or passing values
  - Introduced in **ES6**

Task	Spread Operator (`...`)	Alternative Method
Copy array	`[...arr]`	`arr.slice()`
Merge arrays	`[...arr1, ...arr2]`	`arr1.concat(arr2)`
Copy object	`{...obj}`	`Object.assign({}, obj)`
Merge objects	`{...obj1, ...obj2}`	`Object.assign({}, obj1, obj2)`

Array

- Arrays are ordered collections of elements
  - Arrays are zero-indexed
  - Values (called elements) can be of any type
- Create an array
    - const arr1 = new Array();
    - const arr2 = Array(10);
    - const arr3 = [element1, element2, element3, ...];
- Properties
  - length
  - Access elements using square brackets [index]

Array Methods

- Add/Remove Elements

Method	Description	Example
`push()`	Adds one or more elements to the **end**	`arr.push(4)`
`pop()`	Removes the **last element**	`arr.pop()`
`unshift()`	Adds one or more elements to the **start**	`arr.unshift(0)`
`shift()`	Removes the **first element**	`arr.shift()`
`splice()`	Adds/removes elements at **specific index**	`arr.splice(1, 2, "a", "b")`

Array Methods

- Access/Copy

Method	Description	Example
`slice()`	Returns a **portion of the array**	`arr.slice(1,3)`
`concat()`	Combines arrays and returns a **new array**	`[1,2].concat([3,4])`
`indexOf()`	Returns **index** of element (first match)	`arr.indexOf(2)`
`lastIndexOf()`	Returns **last index** of element	`arr.lastIndexOf(2)`
`includes()`	Checks if array **contains** element	`arr.includes(3)`

Array Methods

- **Iteration**
  - reduce

Method	Description	Example
`forEach()`	Executes a function for **each element**	`arr.forEach(x => console.log(x))`
`map()`	Returns **new array** with modified values	`arr.map(x => x*2)`
`filter()`	Returns **new array** with elements passing a condition	`arr.filter(x => x>2)`
`reduce()`	Reduces array to **single value**	`arr.reduce((acc, x) => acc+x, 0)`
`find()`	Returns **first element** matching condition	`arr.find(x => x>2)`
`findIndex()`	Returns **index** of first element matching condition	`arr.findIndex(x => x>2)`

Array Methods

- Sorting & Reversing

Method	Description	Example
`sort()`	Sorts array (alphabetically by default)	`[3,1,2].sort()` → `[1,2,3]`
`reverse()`	Reverses array	`[1,2,3].reverse()` → `[3,2,1]`

Array Methods

- Others

Method	Description
`join(separator)`	Converts array to string, separated by `separator`
`fill(value, start, end)`	Fills array with value from start to end
`some()`	Returns `true` if **any element** passes a test
`every()`	Returns `true` if **all elements** pass a test
`flat()`	Flattens nested arrays (default 1 level)
`flatMap()`	Maps and flattens in one step

Map

- A Map is a collection of key-value pairs, introduced in **ES6**
  - Keys can be any type (objects, functions, primitives).
  - Maintains the insertion order of keys.

Method	Description	Example
`set(key, value)`	Adds or updates a key-value pair in the Map	`map.set("name", "Alice")`
`get(key)`	Returns the value associated with the key, or `undefined` if not found	`map.get("name") // "Alice"`
`has(key)`	Returns `true` if the Map contains the key	`map.has("age") // true`
`delete(key)`	Removes the key-value pair from the Map	`map.delete("age")`
`clear()`	Removes all key-value pairs from the Map	`map.clear()`
`size`	Property that returns the number of key-value pairs in the Map	`map.size // 3`
`keys()`	Returns an iterable of keys	`for (let key of map.keys()) console.log(key)`
`values()`	Returns an iterable of values	`for (let value of map.values()) console.log(value)`
`entries()`	Returns an iterable of `[key, value]` pairs	`for (let [k, v] of map.entries()) console.log(k, v)`
`forEach(callback)`	Executes a callback function once for each key-value pair in insertion order	`map.forEach((v,k) => console.log(k,v))`

Object vs Map

Feature	Object	Map
Key types	Strings or Symbols only	Any value (string, number, object, function, etc.)
Order of keys	Not guaranteed (insertion order mostly preserved in modern JS for strings)	Insertion order is guaranteed
Size	No built-in property	`.size` property available
Iteration	Use `for...in`, `Object.keys()`, `Object.entries()`	Can use `for...of` directly on Map or `.forEach()`
Performance	Slower for frequent additions/removals	Faster for frequent additions/removals
Default keys	Has a prototype chain (`toString`, etc.)	No default keys; keys are clean
Serialization	Can be serialized with `JSON.stringify()`	Cannot directly serialize; convert to array first
Methods for manipulation	Manual using `delete`, `Object.keys()`	Built-in methods: `.set()`, `.get()`, `.delete()`, `.has()`, `.clear()`

Set

- A Set is a collection of unique values
  - Introduced in **ES6**
  - Duplicates are automatically removed
  - Maintains insertion order

Method	Description	Example
`add(value)`	Adds a value to the Set	`set.add(5)`
`delete(value)`	Removes a value from the Set	`set.delete(2)`
`has(value)`	Checks if a value exists in the Set	`set.has(1) // true`
`clear()`	Removes all values	`set.clear()`
`values()`	Returns an iterable of values	`for (let v of set.values()) console.log(v)`
`keys()`	Same as `values()` (for compatibility with Map)	`for (let k of set.keys()) console.log(k)`
`entries()`	Returns `[value, value]` pairs	`for (let [v1,v2] of set.entries()) console.log(v1,v2)`
`forEach(callback)`	Executes callback for each value	`set.forEach(v => console.log(v))`

Array vs Set

Feature	Array	Set
Duplicates	Allows duplicates	Only stores **unique values**
Order	Maintains insertion order	Maintains insertion order
Access by index	Yes (`arr[0]`)	No (use iteration or `for...of`)
Methods	`push` , `pop` , `shift` , `unshift` , `map` , `filter` , `reduce` , etc.	`add` , `delete` , `has` , `clear` , `forEach`
Length/Size	`.length`	`.size`
Performance	Slower for checking existence (`includes`)	Faster for checking existence (`has`)
Use case	Ordered collections, duplicates allowed	Unique values, fast existence checks

Additional materials

- https://www.w3schools.com/js/

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.**

Group Info

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

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)
- 上课时可能会随机抽查前一天的作业, 现场share屏幕讲解, be prepared

Homework - Short Answer Questions

1. What is ECMAScript?
2. What is REPL?
3. What are primitive data types & reference data types in JS?
4. What is immutability? What data types in JS are immutable?
5. What are some examples of falsy values in JS?
6. What is pass by value and pass by reference?
7. What is the difference between slice and splice
8. What are 3 ways to iterate an array? What is their syntax?
9. How do you sort an array of numbers in ascending and descending order?
10. How would you flatten a nested array in JavaScript?

Homework - Short Answer Questions

11. What's the difference between array map and forEach?

12. Explain what is logged in the following:

console.log(x); var x=2;

console.log(x); let x=2;

console.log(x); const x=2;

13. What's Short-circuiting?

14. What is hoisting?

15. What are scopes in JS? What is the difference between var, let and const?

16. What makes a set different from an array?

17. What makes a map different from an object?

18. Explain Template Literal

Homework - Coding Questions

1. Given the string, implement a function to replace the “-” character with space, and remove “extra” space in the string and convert the string to all lowercase.

   const string = " Perhaps The Easiest-to-understand Case For Reduce Is To Return The Sum Of All The Elements In An Array "

Homework - Coding Questions

1. Given the string, implement a function to replace the “-” character with space, and remove “extra” space in the string and convert the string to all lowercase.

   const string = " Perhaps The Easiest-to-understand Case For Reduce Is To Return The Sum Of All The Elements In An Array ";

2. What does this print and why?

```
const result = null || 'default';
console.log(result);
const andResult = true && 'value';
console.log(andResult);
```

Homework - Coding Questions

3. Use slice to copy part of an array:

```
const arr1 = [10, 20, 30, 40];
//write your code here
expected output: [20, 30]
```

4. Use splice to remove 2 elements from index 1:

```
const arr1 = [1, 2, 3, 4];
//write your code here
expected output: [1, 4]
```

5. Leetcode: 1, 217, 268