Summary

▼ SECTION 5: BOOLEAN

▼ **26.Boolean Overview**

- Boolean is a `primitive` data type and the value is just `false` or `true`
- Things need to know about boolean
  - Type conversion
  - Truthy vs Falsy
  - Logical operators
  - Comparison operators
  - If ... Else
  - Switch ... Case

> Reference: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference

▼ **27.Type Conversion**

1.Type conversion vs Type coercion

`Type coercion` is the automatic or implicit conversion of values from one data type to another (such as strings to numbers).

`Type conversion` is similar to type coercion because they both convert values from one data type to another with one key difference — `type coercion` is implicit whereas `type conversion` can be either implicit or explicit.

> Src: https://developer.mozilla.org/en-US/docs/Glossary/Type_coercion

```
// Type coersion
const a = 1 + '2'; // '12' as it auto convert from number 1 to string '1' before doing the addition

const b = 1 - '2' // -1 as it auto convert string '2' to number 2 before doing the substraction

const message = 'JS is easy'
if (message) { // message here will be converted into boolean automatically
  console.log('Really!? 🤣 ')
}
// Explicit conversion
const error = 'Something wrong!';
const hasError = Boolean(error); // we ask to convert string error to boolean

const mark = 9;
const markString = mark.toString(); // or String(mark)

const type = '1';
const typeId = Number.parseInt(type); // or Number(type)
```

REFERENCES
https://developer.mozilla.org/en-US/docs/Glossary/Type_coercion
https://developer.mozilla.org/en-US/docs/Glossary/Type_Conversion
https://stackoverflow.com/questions/8857763/what-is-the-difference-between-casting-and-coercing

2. Three popular type conversion

#	Name	Type Coersion happens when	Explicit Conversion
1	String Conversion	output something or string concatenate	`.toString()` or String()
2	Numeric Conversion	math operations	`parseInt()`, `parseFloat()` or Number()
3	Boolean Conversion	logical operations	`!` operator or Boolean()

> Reference: https://javascript.info/type-conversions

▼ **28.Truthy vs Falsy**

`Truthy` when covert to boolean have the value `true`

`Falsy` when covert to boolean have the value `false`

Value	Description
`false`	The keyword `false`.
`0`	The `Number` zero (so, also `0.0`, etc., and `0x0`).
`-0`	The `Number` negative zero (so, also `-0.0`, etc., and `-0x0`).
`0n`	The `BigInt` zero (so, also `0x0n`). Note that there is no `BigInt` negative zero — the negation of `0n` is `0n`.
`""`, `''`, ``` `` ```	Empty string value.
null	null — the absence of any value.
undefined	undefined — the primitive value.
NaN	NaN — not a number.
`document.all`	Objects are falsy if and only if they have the [[IsHTMLDDA]] internal slot.That slot only exists in `document.all` and cannot be set using JavaScript.

🌐 **REFERENCES**
https://developer.mozilla.org/en-US/docs/Glossary/Truthy
https://developer.mozilla.org/en-US/docs/Glossary/Falsy

▼ **29.Logical Operators**

Basically, having three logical operators: `AND`, `OR`, `NOT`

#	Operator	Desc
1	&&	AND
2	||	OR
3	!	NOT
4	!!	NOT NOT 😅

`AND` table

RULE: ONLY TRUE IF BOTH TRUE

#	A	B	A && B
1	TRUE	TRUE	TRUE
2	TRUE	FALSE	FALSE
3	FALSE	TRUE	FALSE
4	FALSE	FALSE	FALSE

`OR` table

RULE: ONLY FALSE IF BOTH FALSE

#	A	B	A || B
1	TRUE	TRUE	TRUE
2	TRUE	FALSE	TRUE
3	FALSE	TRUE	TRUE
4	FALSE	FALSE	FALSE

`NOT` table

Prefer to use `Boolean(A)` instead of !!A

#	A	!A	!!A
1	TRUE	FALSE	TRUE
2	FALSE	TRUE	FALSE

▼ **30.Comparison Operators**

1. Comparison operators in JS

#	Operator	Desc
1	>, <	greater/less than
2	>=, <=	greater/less than or equal
3	==	equal (check value after type coersion)
4	!=	not equal
5	===	strict equality (check both type and value, type first and value after type coersion)
6	!==	strict non-equality

2. Same data type comparison

- **Number**

```
// Greater than / Less than
1 > 2; // false
1 < 2; // true

// Greater than or equal / Less than or equal
1 >= 2; // false
2 <= 2; // true

// Equal / Not equal
1 == 2; // false
1 != 2; // true
```

- **String**

Strings also have all the numeric comparison operators, but the way of comparing strings is slightly different.
⇒ Compare `character by character`, `left to right`
If the first character is larger, then the string is larger.
If the first character is smaller, then the string is smaller.
If the first character is the same, then go to check the next character.
Continue until the string runs out.

> How do you know which character is bigger? ⇒ https://asecuritysite.com/coding/asc2

```
'a' > 'b' // false
'aa' <= 'b' // true (longer doesn't mean will be greater)

'abc' > 'abd' // false (because c is less than d)
'a' > 'A' // true
'A' > '9' // true

'easy' == 'easy' // true
'easy frontend' != 'easy' // true
'easy frontend' > 'easy' // true
```

- **Boolean**

Boolean values: true / false
`true` is greater than `false`

```
true > false // true
true == true // true
true == false // false
true != false // true
```

3. Different data type comparison

When comparing different data types, js will `automatically convert` the value to `number` for comparison.

```
123 == '123' // true because '123' will be converted to 123
123 == '0123' // true because '01' will be converted to 1

false == 0 // true because false will be converted to 0
true == 1 // true because true will be converted to 1

'0' == false // true
'2' > true // ???
'123' >= 100 // ???
```

> Reference: https://getify.github.io/coercions-grid/

4. Strict equality (=== and ! ==)

Keep the same value (do not automatically convert the data type) and compare with each other.

- If different data type, immediately return `false`
- If the data type is the same, then compare in the same way as mentioned in the comparison of the same data type.

```
111 === '111' // false because they are different data types
0 === false // false
'' === 0 // false

1 === 1 // true
'abc' === 'abc' // true
```

💡 Should use === instead of == to limit unnecessary risks. If you want to use ==, you have to understand what you're really doing

5. Compare with null/undefined

```
// non-strict ==, null and undefined equal but not any other value.
null == undefined; // true

// strict check ===
null === undefined; // false because they are different type

// other comparisons
// null will converted to 0
// undefined will converted to NaN
null > 0 // false
null == 0 // false
null >= 0 // true WHAT???
```

🌐 **REFERENCES**
https://javascript.info/comparison
https://dorey.github.io/JavaScript-Equality-Table/unified/
https://getify.github.io/coercions-grid/

▼ 31.Control flow with if...else

- `condition` is a boolean value
- If `condition` is another value type, it will automatically convert to `boolean`
- Limit the use of else, leave it with if...return
- Limit the use of nested if...else

```
// GOOD
if (condition) doSomething();

// GOOD
if (condition) {
  doSomething();
  doSomethingElse();
}

// OK
if (condition) {
  doSomething();
} else {
  doSomethingElse();
}

// BAD
if (condition) {
  doSomething();
} else if (anotherCondition) {
  doSomethingElse();
} else {
  doSomethingFinally();
}

// Avoid to use nested if
if (condition1) {
  if (condition2) {
    doSomething();
  } else {
    doSomethingElse();
  }
}
```

> Reference: https://javascript.info/ifelse

▼ 32.Control flow with switch...case

Example: Write a function that takes a positive integer between 1 and 12, and returns the string with the name of the corresponding month.
`printMonth(1) --> Jan`

```
// using if...else
function printMonth(n) {
  let month = '';

  if (n === 1) month = 'Jan'
  else if (n === 2) month = 'Feb'
  else if (n === 3) month = 'Mar'
  else if (n === 4) month = 'Apr'
  else if (n === 5) month = 'May'
  else if (n === 6) month = 'Jun'
  else if (n === 7) month = 'Jul'
  else if (n === 8) month = 'Aug'
```

```
  else if (n === 9) month = 'Sep'
  else if (n === 10) month = 'Oct'
  else if (n === 11) month = 'Nov'
  else if (n === 12) month = 'Dec'
  else month = 'Invalid number'

  return month;
}

// using switch...case
function printMonth(n) {
  let month = '';

    switch (n) {
    case 1: {
      month = 'Jan';
      break;
    }
    case 2: {
      month = 'Feb';
      break;
    }
    case 3: {
      month = 'Mar';
      break;
    }
    // ... similar for month 4 -> 11
    case 12: {
      month = 'Dec';
      break;
    }
    default: {
      month = 'Invalid number'
    }
  }
  return month;
}
```

💡 Note:
- If the condition is a specified set of values, use switch...case instead of if...else
- If the condition is an interval (greater/less than) then if...else should be used

▼ SECTION 6: NUMBER

▼ **37.Number Overview**

- Have 2 type of number
  - Integer (`int`): 1, 2, 3, ...
  - Float (`float`): 1.5, 2.5, 3.7, ...). Use periods to separate decimals.
- Note: Javascript just have only one data type is `number` using for integer and float
- Number in JS is saved with `double-precision 64-bit` binary format IEE 754

```
const count = 1;
const mark = 9.5;

const long = 1000000;
const longer = 1_000_000; // rare usage
const short = 1e6; // usually see this

const smaller = 0.0001;
const smallerShort = 1e-4;
```

1.Arithmetic operators

```
const sum = 1 + 2; // 3
const substract = 1 - 2; // -1
const multiply = 1 * 2; // 2
const divide = 1 / 4; // 0.25

const remainder = 7 % 5; // 2 because 7 = 5x1 + 2 (remainder)

// increment
let count = 1;
const n = count++; // 1 postfix, return first, then increase

let count = 1;
const n = ++count; // 2 prefix, increase first, then return

// decrement
let count = 1;
const n = count--; // 1 postfix, return first, then decrease

let count = 1;
const n = --count; // 0 prefix, decrease first, then return

// unary plus: attempts to convert to number if it's not a number
const n = +'123'; // 123

// unary negation: return the negation of the operand
const n = -'123'; // -123

// exponentiation operator
const n = 2 ** 3; // means 2^3 = 8
```

2.Assignment operators

```
let n = 1;
n += 1; // 2, similar to n = n + 1

// subtraction assignment
let n = 1;
n -= 1; // 0 similar to n = n - 1

// multiplication assignment
let n = 1;
n *= 3; // 3 similar to n = n * 3

// division assignment
let n = 1;
n /= 2; // 0.5 similar to n = n / 2
```

2.toString(base)

```
const n = 20;
n.toString(); // '20' (default is 10)
n.toString(10); // '20' Decimal
n.toString(16); // '14' Hexadecimal
n.toString(8); // '24' Octal
n.toString(2); // '10100' Binary
```

🌐 **REFERENCES**

http://steve.hollasch.net/cgindex/coding/ieeefloat.html
https://javascript.info/number
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Expressions_and_Operators

▼ **38.Built-in Object: Number**

1.Use Number as a function

> When used as a function, Number(value) converts a string or other value to the Number type. If the
> value can't be converted, it returns NaN.

```
Number(123); // 123
Number('123'); // 123
Number('abc'); // NaN
```

2.Static properties

#	Prop	Value	Desc
1	Number.EPSILON	2.220446049250313e-16	Độ chênh lệch nhỏ nhất giữa 2 số
2	Number.MIN_VALUE	5e-324	Số dương nhỏ nhất (gần tới số 0)
3	Number.MAX_VALUE	1.7976931348623157e+308	Số dương lớn nhất
4	Number.MIN_SAFE_INTEGER	-9007199254740991	2^53 - 1
5	Number.MAX_SAFE_INTEGER	9007199254740991	2^53 - 1
6	Number.NaN	NaN	Not a Number
7	Number.NEGATIVE_INFINITY	-Infinity	Âm vô cùng
8	Number.POSITIVE_INFINITY	Infinity	Dương vô cùng

> Src: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number#static_properties

3.Convert to numbers (parseInt and parseFloat)

```
Number('123'); // 123
Number.parseInt('1.5'); // 1
Number.parseFloat('1.5') // 1.5

Number('123.5a'); // NaN
Number.parseInt('123.5a'); // 123
Number.parseFloat('123.5a') // 123.5

Number(null); // 0
Number(undefined); // NaN
```

```
Number(true); // 1
Number(false); // 0
```

> Src: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number#static_methods

4.toFixed() and toPrecision()

- toFixed() and toPrecision() both convert from number to string
- toFixed(digits) then fixed the number of digits after the period.
- toPrecision(digits) then rounds to significant digits.

```
const n = 123.525;
n.toFixed(); // 124 as default value for arg is 0
n.toFixed(0); // 124

n.toFixed(1); // 123.5
n.toFixed(2); // 123.53
n.toFixed(3); // 123.525
n.toFixed(4); // 123.5250
n.toFixed(5); // 123.52500
```

```
const n = 123.525;
n.toPrecision(); // 123.525 similar to n.toString()
n.toString(); // 123.525

n.toPrecision(0); // error argument must be between 1 and 100
n.toPrecision(1); // 100
n.toPrecision(2); // 120
n.toPrecision(3); // 124
n.toPrecision(4); // 123.5
n.toPrecision(5); // 123.53
n.toPrecision(6); // 123.525
n.toPrecision(7); // 123.5250
n.toPrecision(8); // 123.52500
```

▼ 39.Built-in Object: Math

> Math is a built-in object that has properties and methods for mathematical constants and functions. It’s not a function object.

1.Popular props

#	Prop	Desc
1	Math.PI	Giá trị của PI, khoảng 3.14159
2	Math.SQRT2	Giá trị của căn bậc 2 của 2 ~ 1.414

2.Popular methos

#	Method	Desc
1	Math.ceil(x)	Làm tròn lên số nguyên gần nhất
2	Math.floor(x)	Làm tròn xuống số nguyên gần nhất
3	Math.round(x)	Làm tròn tới số nguyên gần nhất
4	Math.trunc(x)	Hàm lấy phần nguyên, bỏ phần thập phân
5	Math.random()	Random số thực từ 0 --> 1
6	Math.abs(x)	Lấy giá trị tuyệt đối
7	Math.pow(x, y)	Hàm luỹ thừa, x^y
8	Math.sqrt(x)	Hàm căn bậc 2

```
Math.abs(-5); // 5
Math.abs(5); // 5

Math.pow(2, 3); // 8
Math.pow(10, 2); // 100

Math.sqrt(16); // 4
Math.sqrt(81); // 9
```

> Src: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math

▼ **40.Imprecise Calculation**

```
0.3 === 0.1 + 0.2; // true or false ???

console.log((0.1).toFixed(20));
console.log((0.2).toFixed(20));
console.log((0.1 + 0.2).toFixed(20));
console.log((0.3).toFixed(20));

console.log(0.1 + 0.2 == 0.3);
console.log(0.1 + 0.2 === 0.3);
console.log(Math.abs(0.1 + 0.2 - 0.3) < Number.EPSILON);
console.log((0.1 + 0.2).toFixed(1) === (0.3).toFixed(1));
```

💡 - Not all numbers are stored correctly in Javascript (actually this happens in other languages because of the common data storage standard)
- When working with real numbers, this should be kept in mind for proper handling.
- For comparisons, use the constant EPSILON or toFixed() to round the number before comparing.

> Reference: https://javascript.info/number#imprecise-calculations

▼ **41.Rounding**

#	Method	Desc
1	Math.ceil(x)	Làm tròn lên số nguyên gần nhất
2	Math.floor(x)	Làm tròn xuống số nguyên gần nhất
3	Math.round(x)	Làm tròn tới số nguyên gần nhất
4	Math.trunc(x)	Hàm lấy phần nguyên, bỏ phần thập phân

Value	Math.floor	Math.ceil	Math.round	Math.trunc
3.2	3	4	3	3
2.5	2	3	3	2
-1.1	-2	-1	-1	-1
-1.6	-2	-1	-2	-1

> Reference: https://javascript.info/number#rounding

▼ **42.Random Number**

1. Random in range [0, n]

```
function randomNumber(n) {
  if (n <= 0) return -1;

  const random = Math.random() * n;
  return Math.round(random);
}
```

2. Random in range [a, b]

```
// 2. Random a number in range of [a, b] with a < b
// min: a
// range: b - a
// [10, 100]
// 10
// 100 - 10 = 90
function randomNumberInRange(a, b) {
  if (a >= b) return -1;

  const random = Math.random() * (b - a);
  return Math.round(random) + a;
}
```

▼ SECTION 7: STRING

▼ **48.String Overview**

1.Quotes

- `Single` or double quoted
- Using backticks for formating string, span multiple lines
- String in JS is case-sensitive

```
const name = "Hin";
const name = 'Should use single quote!'; // recommend
const formatName = `My name is ${name} ${1 + 2}`;
```

- Escape ' " back slash

```
const name = 'I\'m a developer';
```

2.Basic

- Javascript does not have a data type for each character, only a common data type is string
- String in javascript uses character encoding UTF-16
- Other character encodings such as: ASCII (7 bits), UFT-8 (8 bits), UTF-16 (16 bits), UTF-32 (32 bits)

🌐 **REFERENCES**
https://stackoverflow.com/questions/2241348/what-is-unicode-utf-8-utf-16
https://en.wikipedia.org/wiki/Character_encoding
https://en.wikipedia.org/wiki/UTF-16
https://asecuritysite.com/coding/asc2

3.String is immutable

```
let name = 'Easy Frontend';
// name[0] = 'e';
// name = 'easy Frontend'
console.log(name);
```

> Reference: https://javascript.info/string

▼ 49.String - A primitive wrapper object

1.Instance Properties

```
const name = 'Easy Frontend';
name.length; // 13 (read-only)
```

2.Instance Methods

#	Name	Desc
1	charAt(index)	Lấy ký tự tại vị trí `index`
2	concat(str [, ...strN])	Nối chuỗi
3	includes(searchString [, position])	Có chứa chuỗi nào đó hk?
4	startsWith(searchString [, length])	Có bắt đầu với chuỗi `searchString`?
5	endsWith(searchString [, length])	Có kết thúc bằng chuỗi `searchString`?
6	indexOf(searchValue [, fromIndex])	Vị trí đầu tiên có `searchValue`
7	lastIndexOf(searchValue [, fromIndex])	Vị trí cuối cùng có `searchValue`
8	match(regexp)	Liên quan tới regular expresion (tạm bỏ qua)
9	matchAll(regexp)	Liên quan tới regular expresion (tạm bỏ qua)
10	padStart(targetLength [, padString])	Thêm vào đầu string
11	padEnd(targetLength [, padString])	Thêm vào cuối string
12	repeat(count)	Nhân chuỗi hiện tại lên `count` lần
13	replace(searchFor, replaceWith)	Thay thế chuỗi `searchFor` đầu tiên bằng `replaceWith`
14	replaceAll(searchFor, replaceWith)	Thay thế tất cả chuỗi `searchFor` bằng `replaceWith`
15	slice(beginIndex [, endIndex])	Lấy chuỗi con
16	substring(indexStart [, indexEnd])	Lấy chuỗi con
17	split([sep [, limit]])	Tách chuỗi thành mảng các chuỗi con
18	trim()	Bỏ khoảng trắng đầu + cuối string
19	trimStart()	Bỏ khoảng trắng đầu string
20	trimEnd()	Bỏ khoảng trắng cuối string
21	toLowerCase()	Chuyển chuỗi thành chữ viết thường
22	toUpperCase()	Chuyển chuỗi thành chữ viết hoa

```
'Easy Frontend'.charAt(0); // E
'Easy Frontend'.charAt(3); // y

const name = 'Easy Frontend';
name.charAt(name.length - 1); // d (the last character)

''.concat('Easy'); // 'Easy'
''.concat('Easy', ' ', 'Frontend'); // 'Easy Frontend'

'a'.padStart(5, '*'); // ****a
'ab'.padStart(5, '*'); // ***ab
'abc'.padStart(5, '*'); // **abc

'a'.padEnd(5, '*'); // a****
'ab'.padEnd(5, '*'); // ab***
'abc'.padEnd(5, '*'); // abc**

'*'.repeat(5); // *****

' Easy Frontend '.trim(); // 'Easy Frontend'
' Easy Frontend '.trimStart(); // 'Easy Frontend '
' Easy Frontend '.trimEnd(); // ' Easy Frontend'
```

🌐 **REFERENCES**
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String
https://javascript.info/string

▼ **50.Common Operation**

1.Get the character at index

```
'Easy Frontend'.charAt(0); // E
'Easy Frontend'.charAt(3); // y

const name = 'Easy Frontend';
name.charAt(name.length - 1); // d (the last character)
```

2.String concatenation

```
const a = 'Easy';
const b = 'Frontend';

const c = a + ' ' + b; // 'Easy Frontend'
const d = a.concat(' ', b); // 'Easy Frontend'
const e = `${a} ${b}` // 'Easy Frontend' (recommended)
```

3.String traversal

```
const name = 'Easy Frontend';
  for (let i = 0; i < name.length; i++) {
    console.log(name[i])
}
```

4.Switch to lowercase

```
'Easy Frontend'.toLowerCase(); // easy frontend
'Easy Frontend'.toUpperCase(); // EASY FRONTEND
```

5.Find sub-string

```
const name = 'Hello Po';

name.indexOf('o'); // 4
name.lastIndexOf('o'); // 7
```

6.Check contain sub-string

```
const name = 'Easy and Frontend';

name.startsWith('easy'); // false
name.startsWith('Easy'); // true
```

```
name.startsWith('and'); // false

name.includes('Easy'); // true
name.includes('and'); // true
name.includes('Frontend'); // true

name.endsWith('Frontend'); // true
name.endsWith('and'); // false
name.endsWith('frontend'); // false
```

🌐 **REFERENCES**
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String
https://javascript.info/string

▼ **51.Substring**

- Use 1 of 3 functions is okay. But to avoid confusion, remember one function to use is slice
- `substr` then according to MDN it is marked as deprecated
- slice and substring use quite similar
- `slice` supports negative numbers. Negative numbers are counted from the end of the string
- `substring` then consider negative numbers as 0
- `substring` allows start > end, while slice does not

#	Name	Desc
1	slice(start, end)	negative means count from end, start can't be greater than end
2	substring(start, end)	negative means 0, start can be greater than end
3	substr	deprecated as in MDN docs

```
// slice
'Easy Frontend'.slice(0, 4); // 'Easy'
'Easy Frontend'.slice(2); // 'sy Frontend'

'Easy Frontend'.slice(-3); // 'end'
'Easy Frontend'.slice(-3, -1); // 'en'

// substring
'Easy Frontend'.substring(0, 4); // 'Easy'
'Easy Frontend'.substring(2); // 'sy Frontend'

'Easy Frontend'.substring(-3); // 'Easy Frontend'
'Easy Frontend'.substring(-3, -1); // ''

// slice vs substring
'Easy Frontend'.slice(4, 0); // ''
'Easy Frontend'.substring(4, 0); // 'Easy'
```

🌐 **REFERENCES**
https://javascript.info/string#getting-a-substring
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String

▼ 52.Replace

#	Name	Desc
1	`replace`(searchFor, replaceWith)	Tìm kiếm và thay thế một `searchFor` bởi `replaceWith`
2	`replaceAll`(searchFor, replaceWith)	Tìm kiếm và thay thế tất cả `searchFor` bởi `replaceWith`

```
'easy frontend'.replace(' ', '-'); // easy-frontend
'easy frontend'.replace(' ', ''); // easyfrontend

'easy frontend'.replace('easy', 'Easy'); // Easy frontend
'easy frontend'.replace('easy', ''); // ' frontend'

// replace the first match only
'easy easy frontend'.replace('easy', ''); // ' easy frontend'

'learn easy frontend'.replaceAll(' ', '-'); // 'learn-easy-frontend'
'learn easy frontend easy'.replaceAll('easy', ''); // 'learn frontend '

// replace with regexp (Regular Expression)
'super 123 cool'.replaceAll(/[0-9]/gi, '')
```

🌐 **REFERENCES**
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String

▼ 53.Split and Join

string to array: `split()`
array to string: `join()`

1. `split()`

⇒ The function used to split the current string into an array of substrings.

```
'easy'.split(); // ['easy']
'easy'.split(' '); // ['easy']
'easy'.split('-'); // ['easy']
'easy'.split('as'); // ['e', 'y']

'easy'.split(''); // ['e', 'a', 's', 'y']

// count words
const name = 'Easy Frontend';
```

```
const words = name.splits(' '); // ['Easy', 'Frontend']
console.log(words.length); // 2
```

2. `join()`

⇒ The function is used to combine the items of an array into a string.

```
const words = ['Easy', 'Frontend'];
words.join(); // 'Easy,Frontend' as default separator is comma
words.join(''); // 'EasyFrontend'
words.join(' '); // 'Easy Frontend'
words.join('-'); // 'Easy-Frontend'
```

3.Example

```
// CAPITALIZE
function capitalize(str) {
  if (str === '') return '';

  const firstLetter = str[0].toUpperCase();
  const rest = str.slice(1).toLowerCase();

  return `${firstLetter}${rest}`;
}
console.log(capitalize(''));
console.log(capitalize('ABC'));
console.log(capitalize('EasY FroneND'));

//-------------------------------------------------------
// CHECK INCLUDE
// indexOf
// lastIndexOf
// includes

function hasEmail(str) {
  return str.includes('@gmail.com');
  // return str.indexOf('@gmail.com') >= 0;
  // return str.lastIndexOf('@gmail.com') >= 0;
}

console.log(hasEmail('abc abc@gmail.com def'));
console.log(hasEmail('abc abc@gmail def'));

//-------------------------------------------------------
// PARAMETERIZE
// toLowerCase()
// p1: replaceAll()
// p2: split and join
// ['code', 'js', 'is', 'fun']
function parameterize(str) {
  // return str.toLowerCase().replaceAll(' ', '-');
  return str.toLowerCase().split(' ').join('-');
}

console.log(parameterize('Code JS Is Fun'));

//-------------------------------------------------------
// TRUNCATE
/*
  Horizontal ellipsis code is 8230. String.fromCodePoint(8230)
  UTF-16 '\u2026'
  https://www.compart.com/en/unicode/U+2026
*/
// Easy --> truncate('Easy Frontend', 4) --> 'Eas…'
// length
// length <= maxLength --> show full
// length > maxLength --> truncate
function truncate(text, maxlength) {
  if (text.length <= maxlength) return text;

  const shortStr = text.slice(0, maxlength - 1);
  return `${shortStr}\u2026`;
}
```

```
console.log(truncate('Easy', 4));
console.log(truncate('Easy Frontend', 4));
```

🌐 **REFERENCES**
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String

▼ SECTION 8: OBJECT

▼ **58.Object Overview**

1.Declare object

- With the data types like number, string, boolean, it's just a simple value
- But with object is a data type that can contain many different data through `key`, `value`
- `value` can be any data type: number, string, boolean, object, array, function, ...

```
object syntax

{
  key1: value1,
  key2: value2,
  ...
}
```

```
const student = {
  id: 1,
  name: 'Van A',
  name: 'Van B', // same key come later will take precedence
  isHero: true,
  'key has space': 'super', // key with spaces should be wrapped in quotes

  sayHi() {
    console.log('Hello!');
  }
}
```

2.Get value of key

- Use the dot operator to access the object's key
- Use square brackets to access an object's dynamic key, including keys with spaces

```
const student = {
  id: 1,
  name: 'Van A',
  isHero: true,

  'avg mark': 9,
}

console.log(student.name);
console.log(student.avg mark); // Syntax Error
console.log(student['avg mark']); // 9

const key = 'avg mark';
```

```
console.log(student.key); // undefined
console.log(student[key]); // 9
```

3.Add new key to object

```
const student = {
  id: 1,
  name: 'Van A',
  isHero: true,
}

// update value of a key
student.name = 'Van B';

// simply set new key for object
student.age = 18;
student['mark'] = 10;

console.log(student.age); // 18
console.log(student.mark); // 10
```

4.Delete a key

- To remove a key from the object, use `delete` operator

```
const student = {
  id: 1,
  name: 'Van A',
  isHero: true,
}

// Remove "name" key
delete student.name;

console.log(student.name); // undefined
```

▼ 59.Value vs Reference

#	Name	Data Type
1	Primitive Type / Value Type	boolean, number, string, null, undefined, symbol
2	**Reference Type**	object, array, function

1.Distinguish value and reference type

```
// value type, the variable store the value
const name = 'Easy Frontend';
const age = 18;
const isHero = true;
const selectedStudent = null;
// |---------------------------|
// | name = 'Easy Frontend' |
// |_____________________________|
```

```
// reference type, the variable store the address/ref of the value
// so in this example
// object value { name, age, ... } will be store at address 123456 (somewhere in memory)
// and the student variable just hold the address 123456 (that's why we call reference)
const student = {
  name: 'Easy Frontend',
  age: 18,
}
// |----------------------|  | ADDRESS: 123456 |
// | student = 123456 | ---> | VALUE: { name, age, ... } |
// |__________________________|  |_____________________________|
```

2.Assignment with object

```
// primitive type
const a = 5;
let b = a;
b = 10;

console.log(a); // 5
```

```
// reference type
const student1 = { name: 'Easy Frontend', };
const student2 = student1;
student2.name = 'Hin Tran';

console.log(student1.name); // Hin Tran ???
```

3.Compare object

- When comparing references (object, array, function), the reference address will be compared. If both point to the same reference address, it will return true, otherwise it will return false

```
const student1 = { name: 'Easy Frontend', };
const student2 = student1; // copy reference from student1 to student2

student1 === student2; // true as they both point to the same reference
```

```
const student1 = { name: 'Easy Frontend', }; // reference 1
const student2 = { name: 'Easy Frontend', }; // reference 2

student1 === student2; // false as they are different refs
```

💡 In case you need to compare two objects, you can consider comparing the key primitive types of the two objects.

4.Pass by value vs Pass by reference

```
function changePrimitive(name, age) {
  name = 'Easy Frontend';
  age = 18;
}

let name = 'Easy';
let age = 17;
changePrimitive(name, age);

console.log(name); // 'Easy'
console.log(age); // 17
```

```
function changeReference(student) {
  student.name = 'Easy Frontend';
  student.age = 18;
}

const student = {
  name: 'Easy',
  age: 17,
}
changeReference(student);

console.log(student.name); // 'Easy Frontend'
console.log(student.age); // 18
```

▼ **60.Common Operation**

1.Name the key

- With variable / function names, reserved keywords cannot be used
- As for the object's key name, it's comfortable, including reserved keywords, but not recommended

```
const student = {
  name: 'Easy Frontend',

  const: 'haha',
  function: 'it works',
  true: 'work too',
}
```

2.Property value shorthand

```
const name = 'Easy Frontend';
const age = 18;

const student = {
  name: name, // key and value variable have the same name
  age: age, // key and value variable have the same name
}

// shorthand (recommended)
const student = {
  name,
  age,
}
```

3.Object destructuring

```
const student = {
  name: 'Easy Frontend',
  age: 18,
}

// old way
const name = student.name;
const age = student.age;

// new way usign object destructuring
const { name, age } = student; // recommended
```

4.Check key in onject?

Use the `in` operator to check if a key exists in the object.

```
const student = {
  name: 'Easy Frontend',
  age: 18,
}

'name' in student // true
'age' in student // true
'isHero' in student // false
```

5.Clone object

```
const student = {
  name: 'Easy Frontend',
  age: 18,
}

const moreProps = {
  isHero: true,
  gender: 'male',
}

// v1: using Object.assign()
const clonedStudent = Object.assign({}, student, moreProps);

// v2: using spread operator (shorter, easier to read)
const clonedStudent2 = {
  ...student,
  ...moreProps,
}
```

6.Deep clone object

```
const student = {
  name: 'Easy Frontend',
  age: 18,

  mark: {
    math: 10,
    english: 7,
  }
}

const clonedStudent = {
  ...student,
}
clonedStudent.mark.math = 1;
console.log(student.mark.math); // 1

// solution, clone nested levels if any
const clonedStudent = {
  ...student,
  mark: {
    ...student.mark,
  }
}
clonedStudent.mark.math = 1;
console.log(student.mark.math); // 10 works now
```

> Reference: https://github.com/immerjs/immer

▼ **61.Traverse keys of object**

- Should use for...in to traverse keys of object

```
const student = {
  id: 1,
  name: 'Van A',
  isHero: true,
}

const keyList = Object.keys(student); // ['id', 'name', 'isHero']
for (let i = 0; i < keyList.length; i++) {
  const key = keyList[i];

  console.log('key:', key); // id, name, isHero
  console.log('value:', student[key]); // 1, 'Van A', true
}

// or a similar way using forEach
Object.keys(student).forEach(key => {
  console.log('key:', key); // id, name, isHero
  console.log('value:', student[key]); // 1, 'Van A', true
})
```

```
const student = {
  id: 1,
  name: 'Van A',
  isHero: true,
}

// recommended
for (let key in student) {
  console.log('key:', key); // id, name, isHero
  console.log('value:', student[key]); // 1, 'Van A', true
}
```

Example

```
// 3. Get average mark of an object
// calcAvgMark({ math: 10, english: 8 }) --> 9
function calcAvgMark(mark) {
  if (!mark) return -1;

  // avg = sum / length
  const length = Object.keys(mark).length;

  let sum = 0;
  for (const key in mark) {
    const value = mark[key];
    sum += value;
  }

  return (sum / length).toFixed(1);
}

console.log(calcAvgMark({ math: 10, english: 7 }));
```

▼ SECTION 9: ARRAY

▼ 63.Array Overview

1.Declare an array

- Domain name, using the suffix `List`. Eg: use `numberList` instead of `number`
- Each element can have a different data type.

```
// declare an empty array
const numberList = [];
```

```
const numberList = [1, 2, 3]; // a list of numbers
const wordList = ['Easy', 'Frontend']; // a list of strings
const flagList = [true, false, false]; // a list of boolean

// a list of students
const studentList = [
  { id: 1, name: 'Van A' },
  { id: 2, name: 'Thi B' },
  { id: 3, name: 'Van C' },
]
```

```
// a list of list
const board = [
  [1, 2],
  ['a', 'b', 'c'],
  [true, false, false, false]
]

// a list of mixed data type
const mixedList = [
  1,
  2,
  'word',
  true,
  null,
  undefined,
  { id: 1, name: 'Easy' },
  [1, 2, 3]
];
```

2.Retrieve an element of an array

- Use square brackets to access the element at the index position
- Index of array starts from 0
- If the length of the array is 3, then the largest index is length - 1 = 2

```
const numberList = [3, 5, 7]; // recommended

numberList[0]; // 3
numberList[1]; // 5
numberList[2]; // 7

numberList.length; // 3
numberList[numberList.length - 1]; // 7 (the last element)
```

3.Two-dimensional arrays

```
const board = [
  [1, 2, 3],
  [4, 5, 6],
  [7, 8, 9],
]

board[0]; // [1, 2, 3]
board[1]; // [4, 5, 6]
board[2]; // [7, 8, 9]

board[0][1]; // 2
board[1][2]; // 6
```

▼ **64.Array Object**

1.Static mothods

#	Name	Desc
1	Array.isArray(arr)	Kiểm tra `arr` có phải là mảng không?
2	Array.from()	Tạo mảng mới từ các dữ liệu khác như Set, Iterable, ...

2.Instance props

```
const numberList = [1, 2, 3];
numberList.length; // 3
```

3.Instance methods

- Check element exists

#	Name	Desc
1	every(callbackFn)	Kiểm tra tất cả phần tử thoả điều kiện
2	some(callbackFn)	Kiểm tra có một phần tử thoả điều kiện
3	indexOf(searchElement)	Tìm vị trí đầu tiên của phần tử `searchElement`
4	lastIndexOf(searchElement)	Tìm vị trí cuối cùng của phần tử `searchElement`
5	includes(searchElement)	Kiểm tra có chứa phần tử `searchElement` không
6	find(callbackFn)	Tìm phần tử đầu tiên thoả điều kiện
7	findIndex(callbackFn)	Tìm vị trí của phần tử đầu tiên thoả điều kiện

- Add/remove element

#	Name	Desc
1	push(element0, ..., elementN)	Thêm cuối mảng
2	pop()	Xoá cuối mảng
3	shift()	Xoá đầu mảng
4	unshift(element0, ..., elementN)	Thêm đầu mảng
5	splice(start, deleteCount, item1, ..., itemN)	Xoá/thêm giữa mảng

- Popular function

#	Name	Desc
1	forEach(callbackFn)	Duyệt mảng
2	map(callbackFn)	Biến đổi mảng
3	filter(callbackFn)	Lọc mảng theo điều kiện
4	slice(start, end)	Lấy mảng con
5	reduce()	Duyệt mảng và tính toán cho ra kết quả cuối cùng

- Other function

#	Name	Desc
1	fill(value, start = 0, end = arr.length)	Fill value từ start tới end
2	join()	Biến đổi mảng thành chuỗi
3	concat()	Nối mảng
4	reverse()	Đảo ngược mảng
5	sort()	Sắp xếp mảng

```
Array.isArray(123); // false
Array.isArray('Easy Frontend'); // false
Array.isArray(true); // false
Array.isArray([]); // true
Array.isArray([1, 2, 3]); // true

[null, undefined].fill(false); // [false, false]
Array(5).fill(1); // [1, 1, 1, 1, 1]
['easy', 'frontend'].join('-'); // 'easy-frontend'
[1, 2, 3].reverse(); // [3, 2, 1]
```

> Reference: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array

▼ **65.Array Destructuring**

```
// object destructuring
const student = {
  id: 1,
  name: 'Easy Frontend',
}

const { id, name } = student;
console.log(id); // 1
console.log(name); // 'Easy Frontend'
```

```
const numberList = [3, 5, 7];

// old way
const first = numberList[0]; // 3
const second = numberList[1]; // 5
const third = numberList[2]; // 7

// similar but new way
const [first, second, third] = numberList;

// or even direct array
const [first, second, third] = [3, 5, 7, 9, 11];

// rest operator (...)
const [first, second, third, ...rest] = [3, 5, 7, 9, 11];
console.log(rest); // [9, 11]
```

▼ 66.Clone Array

Issue: Array is also a reference, so be careful when using assignment.

```
const numberList = [1, 2, 3];
const anotherList = numberList;

anotherList[0] = 4; // anotherList = [4, 2, 3]
console.log(numberList); // [4, 2, 3]
```

Solution: Clone array before making changes.

```
const numberList = [1, 2, 3];
const anotherList = [...numberList];

anotherList[0] = 4; // anotherList = [4, 2, 3]
console.log(numberList); // [1, 2, 3]
```

▼ 67.Loop - Traverse Elements Of Array

- Before ES5: for...i
- ES5: forEach
- ES6: for...of

```
// before ES5
const numberList = [2, 4, 6];
for (let i = 0; i < numberList.length; i++) {
  const number = numberList[i];
  console.log(number); // 2, 4, 6
}
```

```
// ES5 forEach
const numberList = [2, 4, 6];
```

```
numberList.forEach(x => console.log(x)); // 2, 4, 6
```

```
// ES6 for...of
const numberList = [2, 4, 6];
for (const number of numberList) {
  console.log(number); // 2, 4, 6
}
```

▼ 68.Add/Remove Elements Of Array

#	Name	Desc
1	push(element0, ..., elementN)	Thêm cuối mảng
2	pop()	Xoá cuối mảng
3	shift()	Xoá đầu mảng
4	unshift(element0, ..., elementN)	Thêm đầu mảng
5	splice(start, deleteCount, item1, ..., itemN)	Xoá/thêm giữa mảng

```
// Add new items at the end
/*
  The push() method adds one or more elements to the end of an array
  and returns the new length of the array.
*/
const numberList = [1, 2, 3];
numberList.push(4, 5);
console.log(numberList); // [1, 2, 3, 4, 5]
```

```
// Remove items at the end
/*
  The pop() method removes the last element from an array
  and returns that element.
  This method changes the length of the array.
*/
const numberList = [1, 2, 3];
const lastNumber = numberList.pop();
console.log(numberList, lastNumber); // [1, 2], 3
```

```
// Add new items at the beginning
/*
  The unshift() method adds one or more elements to the beginning of
  an array and returns the new length of the array.
*/
const numberList = [1, 2, 3];
numberList.unshift(0);
console.log(numberList); // [0, 1, 2, 3]
```

```
// Remove items at the beginning
/*
  The shift() method removes the first element from an array
  and returns that removed element.
  This method changes the length of the array.
*/
const numberList = [1, 2, 3];
```

```
const firstNumber = numberList.shift();
console.log(numberList, firstNumber); // [2, 3], 1
```

```
// Add/remove items at the middle of an array
const numberList = [3, 5, 7];
numberList.splice(0, 0, 2, 4);
console.log(numberList);
```

```
const numberList = [1, 3, 5, 7]; // [1, 3, 2, 4, 7]
numberList.splice(2, 1, 2, 4);
console.log(numberList);

const numberList = [1, 3, 5, 7]; // [1, 3, 4, 4, 5, 7]
numberList.splice(2, 0, 4, 4);
console.log(numberList);

const numberList = [1, 3, 5, 7];
numberList.pop();
numberList.pop();
console.log(numberList);

// [1, 2, 3]
// [1, 2, 3, empty]
// [0, 1, 2, 3]
```

▼ 69.Check For Element Existence

#	Name	Desc
1	every(callbackFn)	Kiểm tra tất cả phần tử thoả điều kiện
2	some(callbackFn)	Kiểm tra có một phần tử thoả điều kiện
3	indexOf(searchElement)	Tìm vị trí đầu tiên của phần tử `searchElement`
4	lastIndexOf(searchElement)	Tìm vị trí cuối cùng của phần tử `searchElement`
5	includes(searchElement)	Kiểm tra có chứa phần tử `searchElement` không

```
// check if all numbers is even
[1, 2, 4].every((x) => x % 2 === 0); // false
[2, 4, 6].every((x) => x % 2 === 0); // true
```

```
// check if exist one number is even
[1, 2, 4].some((x) => x % 2 === 0); // true
[1, 3, 5].some((x) => x % 2 === 0); // false
```

```
[1, 1, 1].indexOf(1); // 0
[1, 1, 1].lastIndexOf(1); // 2

['easy', 'frontend', 'easy'].indexOf('easy'); // 0
['easy', 'frontend', 'easy'].lastIndexOf('easy'); // 2

['easy', 'frontend', 'easy'].includes('easy'); // true
['frontend'].includes('easy'); // false
```

```
// every v1
function checkIfAllEven(numberList) {
  if (!Array.isArray(numberList)) return false;

  let isValid = true;
```

```
  for (let i = 0; i < numberList.length; i++) {
    const number = numberList[i];
    if (number % 2 !== 0) {
      isValid = false;
      break;
    }
  }
  return isValid;
}

console.log(checkIfAllEven([2, 1, 3]));
console.log(checkIfAllEven([2, 4, 6]));
```

```
// every v2
function checkIfAllEven(numberList) {
  if (!Array.isArray(numberList)) return false;

  for (let i = 0; i < numberList.length; i++) {
    if (numberList[i] % 2 !== 0) return false;
  }

  return true;
}

console.log(checkIfAllEven([2, 1, 3]));
console.log(checkIfAllEven([2, 4, 6]));
```

▼ **70.** `find()`

#	Name	Desc
1	find(callbackFn)	Tìm phần từ đầu tiên thoả điều kiện
2	findIndex(callbackFn)	Tìm vị trí của phần từ đầu tiên thoả điều kiện

```
[2, 1, 3].find(x => x % 2 === 0); // 2
[2, 1, 3].findIndex(x => x % 2 === 0); // 0

['easy', 'frontend'].find(x => x.length > 4); // 'frontend'
['easy', 'frontend'].findIndex(x => x.length > 4); 1
```

> Reference: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/find

1.What is callback?

> A callback function is a function passed into another function as an argument, which is then invoked inside the outer function to complete some kind of routine or action.

```
function main(callbackFn) {
  // processing ...
  // do another stuff
  callbackFn()
}

function callback() {
  console.log('call me when needed')
}

main(callback)
```

```
function main(onFinish) {
  let sum = 0;
  for (let i = 0; i < 10; i++) {
    sum += i;
  }

  onFinish(sum)
}
function handleOnFinish(sum) {
  console.log('Sum is:', sum);
}

main(handleOnFinish);
```

▼ 71. `find()` with for...i

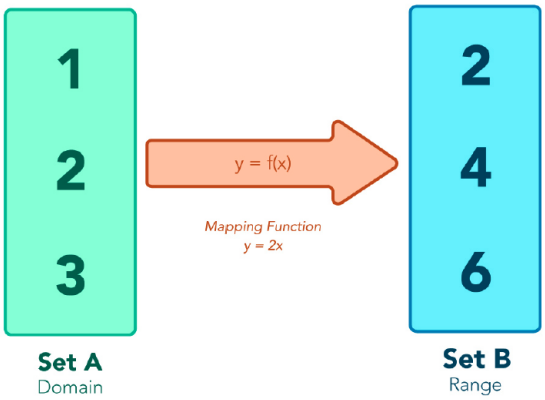
```
// find(callbackFn)
// find(x => x > 0)

// v1
function findFirstEven(numberList) {
  if (!Array.isArray(numberList)) return undefined;

  let firstEven;

  for (let i = 0; i < numberList.length; i++) {
    const number = numberList[i];
    if (number % 2 === 0) {
      firstEven = number;
      break;
    }
  }

  return firstEven;
}

// v2
function findFirstEven(numberList) {
  if (!Array.isArray(numberList)) return undefined;

  for (let i = 0; i < numberList.length; i++) {
    const number = numberList[i];
    if (number % 2 === 0) {
      return number;
    }
  }

  return undefined;
}

// v3
function find1(arr, callbackFn) {
  if (!Array.isArray(arr)) return undefined;

  for (let i = 0; i < arr.length; i++) {
    const element = arr[i];
    if (callbackFn(element, i)) return element;
  }

  return undefined;
}

console.log(find1([1, 2, 3, 5], (number) => number % 2 === 0));
```

```
function findIndex1(arr, callbackFn) {
  if (!Array.isArray(arr)) return undefined;

  for (let i = 0; i < arr.length; i++) {
    const element = arr[i];
    if (callbackFn(element, i)) return i;
  }

  return undefined;
}
```

```
console.log(findIndex1(['a', 'hientran', 'b', 'c'], (letter) => letter.length > 3));
console.log(findIndex1([2, 2, 3, 5], (x) => x % 2 === 0));
```

▼ **72.** `map()`

- Use the `map(transformFn)` function to transform one element to another
- Note the number of elements does not change
- What's changed here is that each element will be transformed according to the same formula
- The return result is a new array



> Reference: https://www.jbakebwa.dev/posts/js-array-map.html

```
const numberList = [1, 3, 5, 7];
numberList.map(x => x + 1); // [2, 4, 6, 8]
numberList.map(x => x * 2); // [ 2, 6, 10, 14]

const wordList = ['easy', 'frontend'];
wordList.map(x => x.length); // [4, 8]
wordList.map(x => `super-${x}`); // ['super-easy', 'super-frontend']
```

```
// map1(arr, callbackFn)

function map1(arr, callbackFn) {
  if (!Array.isArray(arr) || typeof callbackFn !== 'function') return undefined;

  let newArray = [];

  // mapping...
  for (let i = 0; i < arr.length; i++) {
    const element = arr[i];
    const newElement = callbackFn(element, i);
    newArray.push(newElement);
  }

  return newArray;
}

console.log(map1([1, 2, 3], (x) => x + 1));
console.log(map1([1, 2, 3], (x) => x * 2));
console.log(map1([1, 2, 3, 4], (x, index) => (index % 2 === 0 ? x + 1 : x * 2)));
```

> Reference: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/map

▼ 73. `filter()`

- Use `filter(callbackFn)` function to filter array according to given condition
- The returned result is a new subarray

```
const numberList = [1, 3, 5, 2, 7];
numberList.filter(x => x % 2 === 0); // [2]

numberList.filter(x => x > 2); // [3, 5, 7]

numberList.filter(x => x <= 10 || x % 5 === 0); // [5]
```

```
const wordList = ['easy', 'frontend', 'easier'];
wordList.filter(x => x.length < 5); // ['easy']

wordList.filter(x => x.startsWith('ea')); // ['easy', 'easier']
```

```
// filter(arr, callbackFn)
function filter(arr, callbackFn) {
  let newArray = [];

  for (let i = 0; i < arr.length; i++) {
    const element = arr[i];
    if (callbackFn(element, i)) {
      newArray.push(element);
    }
  }

  return newArray;
}

console.log(filter([1, 2, 3, 4, 5], (x) => x > 3));
console.log(filter([1, 2, 3, 4, 5], (x, ixd) => x % 2 === 0 && ixd % 2 !== 0));
```

> Reference: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/filter

▼ 74. `sort()`

- In Javascript, there is built-in `sort(compareFn)` function to sort the array according to the desired condition
- If it is `null/undefined` then auto put to the end of the array, `null` comes before `undefined`
- If the `compareFn` function is not provided, the elements are converted to strings for comparison
- If `compareFn(a, b)` is provided, determine by the result of the function:
  - Result< 0, a will come before b
  - Result = 0, a and b are the same
  - Returns > 0, a will come after b
- The sort() function will return the array after sorting (but this is the current array, not the new array)

```
const numberList = [2, 5, 3, 1];
numberList.sort(); // [1, 2, 3, 5]

[null, 2, 1, 5, 3, undefined, null].sort(); // [1, 2, 3, 5, null, null, undefined]
```

```
// v1
function compareFn(a, b) {
  if (a > b) return 1;
  if (a === b) return 0;
  return -1;
}
[2, 1, 3].sort(compareFn); // 1, 2, 3
```

```
// v2
function compareFn(a, b) {
  return a - b;
}
[2, 1, 3].sort(compareFn); // 1, 2, 3
```

```
// v3
function compareFn(a, b) {
  return a - b;
}
[2, 1, 3].sort((a, b) => a - b); // 1, 2, 3
```

🌐 **REFERENCES**
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/sort
https://www.tutorialspoint.com/which-algorithm-does-the-javascript-arrayhashsort-function-use
https://stackoverflow.com/questions/234683/javascript-array-sort-implementation

▼ 75. `reduce()`

- Use the `reduce()` function when there is a need to iterate through each element and calculate a final result.

```
const numberList = [2, 4, 6];
let sum = 0;

for (let i = 0; i < numberList.length; i++) {
  sum += numberList[i];
}

console.log(sum); // 12

// the same with above but using reduce()
numberList.reduce((sum, number) => sum + number); // 12
```

```
// reduce(arr, callbackFn, initialValue)
// Note:
// accumulator should be an array and callbackFn should be a function
// arr.length = 0 and initialValue = undefined --> throw error
// arr.length = 0 and initialValue != undefined --> return initialValue

function reduce(arr, callbackFn, initialValue) {
  if (!Array.isArray(arr) || typeof callbackFn !== 'function') {
    throw new Error('Invalid parameter');
  }

  if (arr.length === 0) {
    if (initialValue === undefined) {
      throw new Error('Should have initial value when array is empty');
    }

    return initialValue;
  }

  const hasInitialValue = initialValue !== undefined;
  const startIndex = hasInitialValue ? 0 : 1;
  let accumulator = hasInitialValue ? initialValue : arr[0];

  for (let i = startIndex; i < arr.length; i++) {
```

```
    accumulator = callbackFn(accumulator, arr[i], i);
  }

  return accumulator;
}

console.log(reduce([1, 2, 3], (sum, number) => sum + number, undefined));
```

> Reference: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/Reduce

▼ **76.Example**

```
// Array exercises
// 1. Print numbers: 1 -> 10
function printNumbers() {
  for (let i = 1; i <= 10; i++) {
    console.log(i);
  }
}
printNumbers();

// 2. Print even numbers [2, 4, 6, 8, 10]
function printEvenNumbers() {
  for (let i = 2; i <= 10; i++) {
    if (i % 2 === 0) console.log(i);
  }
}

printEvenNumbers();

// 3. Print even numbers but less than n
function printEvenLessNumbers(n) {
  if (n < 2) return;

  for (let i = 2; i <= n; i++) {
    if (i % 2 === 0) console.log(i);
  }
}
printEvenLessNumbers(5);

function printEvenLessNumbersV2(n) {
  if (n < 2) return;

  for (let i = 2; i < n; i += 2) {
    console.log(i);
  }
}

printEvenLessNumbersV2(9);
```

```
// Write a function to find max of an array
// for...i
// forEach
// reduce

function findMaxI(numberList) {
  if (!Array.isArray(numberList) || numberList.length === 0) return undefined;

  let max = numberList[0];
  for (let i = 0; i < numberList.length; i++) {
    if (numberList[i] > max) max = numberList[i];
  }

  return max;
}

console.log(findMaxI([1, 5, 9, 12]));

function findMaxEach(numberList) {
  if (!Array.isArray(numberList) || numberList.length === 0) return undefined;
```

```
  let max = numberList[0];
  numberList.forEach((number) => {
    if (number > max) max = number;
  });

  return max;
}

console.log(findMaxEach([1, 5, 9, 12]));

function findMaxReduce(numberList) {
  if (!Array.isArray(numberList) || numberList.length === 0) return undefined;

  //   return numberList.reduce((max, number) => {
  //     if (number > max) return number;
  //     return max;
  //   });

  return numberList.reduce((max, number) => (number > max ? number : max));
}

console.log(findMaxReduce([1, 5, 9, 12]));
```

```
// Write a function to find longest word
// for...i
// for...each
// reduce

function findLongestWordI(wordList) {
  if (!Array.isArray(wordList) || wordList.length === 0) return undefined;

  let longestWord = wordList[0];
  for (let i = 0; i < wordList.length; i++) {
    const currentWord = wordList[i];

    if (currentWord.length > longestWord.length) longestWord = currentWord;
  }

  return longestWord;
}

function findLongestWordEach(wordList) {
  if (!Array.isArray(wordList) || wordList.length === 0) return undefined;

  let longestWord = wordList[0];
  wordList.forEach((currentWord) => {
    if (currentWord.length > longestWord.length) longestWord = currentWord;
  });

  return longestWord;
}

function findLongestWordReduce(wordList) {
  if (!Array.isArray(wordList) || wordList.length === 0) return undefined;

  return wordList.reduce((longestWord, currentWord) =>
    currentWord.length > longestWord.length ? currentWord : longestWord
  );
}

const word = ['hin', 'hintran0208'];
console.log(findLongestWordI(word));
console.log(findLongestWordEach(word));
console.log(findLongestWordReduce(word));
```

> Reference: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/Reduce

▼ SECTION 10: UNIT TEST

80. Unit Test Overview

- What: test of units of source code.
- Why: cover cases required + easy to maintain / refactor
- When: required on Backend, but just sometimes on frontend
- Who: developers

Some library support Unit Test: link

Jest framework

- Website: https://jestjs.io/
- Jest is a delightful JavaScript Testing Framework with a focus on simplicity.
- It works with projects using: Babel, TypeScript, Node, React, Angular, Vue and more!
- Create React App uses Jest as its test runner.

81. Set up Jest

```
# Install Jest extension
  Jest
  Jest snippets
  Jest runner

# 1. Initialize npm
npm init

# 2. Install jest type dev dependencies
yarn add --dev jest

# 3. Install Babel packages
yarn add --dev babel-jest @babel/core @babel/preset-env

# 4. Add babel.config.js
module.exports = {
presets: [['@babel/preset-env', {targets: {node: 'current'}}]],
};
```

82. Jest file structure

Setup and Teardown

83. Common matchers

Using Matchers

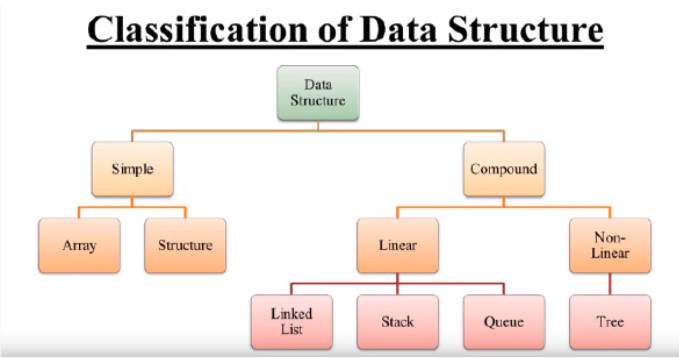
▼ SECTION 11: PRACTISE

▼ SECTION 12: DATA STRUCTURES

▼ **100. Data structure Overview**

- What: defines the structure of a data type to store
- Why: for efficient job processing, each different problem needs a data structure different

- When: data processing
- Who: developers
- Where: more on BE, sometimes on FE



Built-in data types in Javascript

- Primitive Types: number, string, boolean, symbol, null, undefined
- Reference type: object, array, function

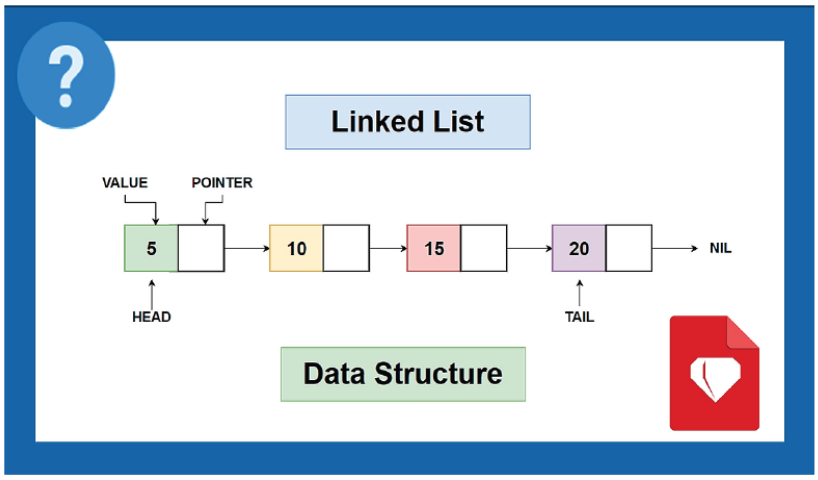
💡 With each Data Structure, we need to care about?

1. Data structure
2. Required operation (Travers through element, search, add, delete, edit, order)
3. How to setup?

▼ 101. Linked List

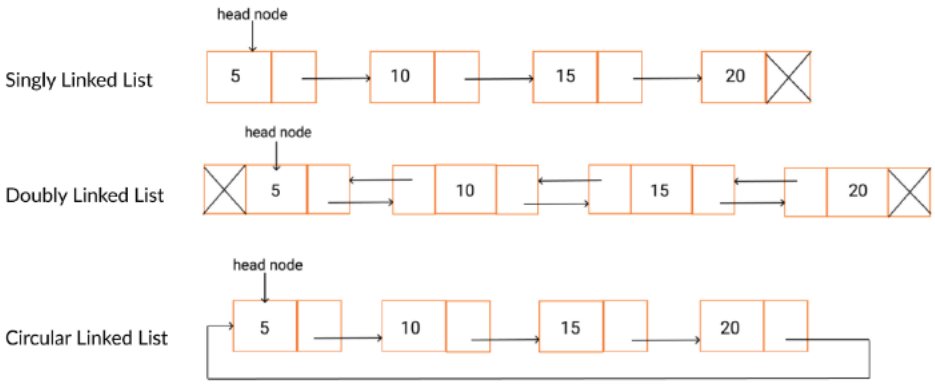
1. What is Linked List?

- Linked list is a linear data structure which is all the elements connect with each other by the links. Each element includes two part: one for data (string, number, object, array...) and one for the connection to the next element.
- The first element is called head
- The last element is called tail (point to the null value to mark this is the last element)




```
const tail = { data: 20, next: null }
const item2 = { data: 15, next: tail }
const item1 = { data: 10, next: item2 }
const head = { data: 5, next: item1 }

{
data: 5,
  next: {
    data: 10,
      next: {
        data: 15,
        next: {
          data: 20,
          next: null,
      }
    }
  }
}
```

2. Types of Linked List



3. Required operations

- Create list
- Traverse all elements

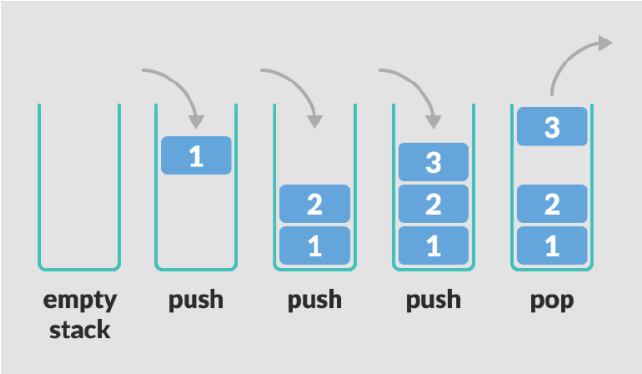
- Search element
- Add: first, middle, last
- Delete: first, middle, last

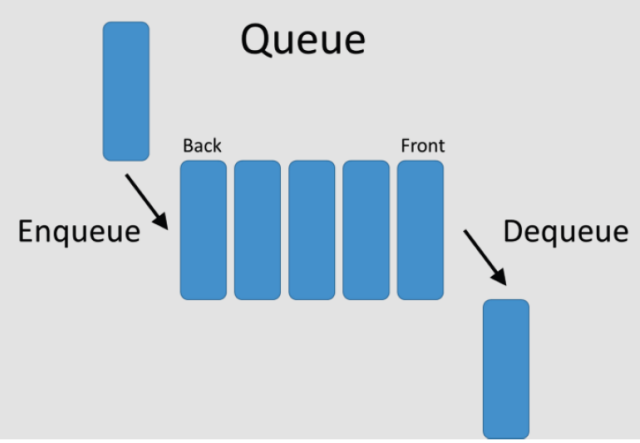
4. When are we using Linked List?

Operation	Array	Linked List
Create	fixed size	dynamic size
Access	Random	In sequence
Search	linear	linear
Add/Remove	not efficient, re-allocate items	easy, no need to re-allocate items
Traversal	linear	linear

⇒ Consider to use Linked List for **large list** with **frequently add/remove operations.**

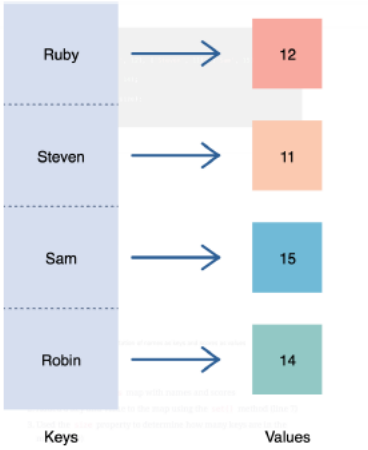
▼ 108. Stack and Queue





	Stack	Queue
Concept	LIFO (Last In First Out)	FIFO (First In First Out)
Pointers	One (top)	Two (front, back/rear)
Add	Push	Enqueue
Remove	Pop	Dequeue
Other operations	isEmpty, getTop, getSize	getFront, getBack, getSize
Random access	NO	NO

▼ **111. JS built-in object: Map**



1. Map Overview

- What: Map is data structures which each data tuple is stored and retrieved a unique key.
- Why: quickly access, don't need to loop through each items like array
- When: easy to access data with key and frequently adding/remove key

Before ES6

```
const cityList = [
  { id: 1, name: 'TP. Hồ Chí Minh' },
  { id: 2, name: 'TP. Phan Thiết' },
]

const studentList = [
  { id: 123, name: 'Alice', cityId: 1 },
  { id: 321, name: 'Bob', cityId: 2 },
]

// Question: how to show city name for each student?
const cityMap = cityList.reduce((map, city) => {
  map[city.id] = city;
  return map;
}, {});

cityMap[1].name; // TP. Hồ Chí Minh
cityMap[2].name; // TP. Phan Thiết
```

2. Map Methods

#	Name	Desc
1	new Map()	Khởi tạo map
2	**size** (thuộc tính, ko phải hàm)	Lấy size của map
3	set(key, value)	Tạo mới / cập nhật value cho key
4	get(key)	Lấy value của key
5	has(key)	Kiểm tra map có key không?
6	delete(key)	Xoá một key
7	clear()	Xoá tất cả keys
8	*keys*()	LOOP qua tất cả keys
9	*values*()	LOOP qua tất cả values
10	*entries*()	LOOP qua tất cả cặp [key, value]
11	*forEach*((value, key) => {})	LOOP qua tất cả values

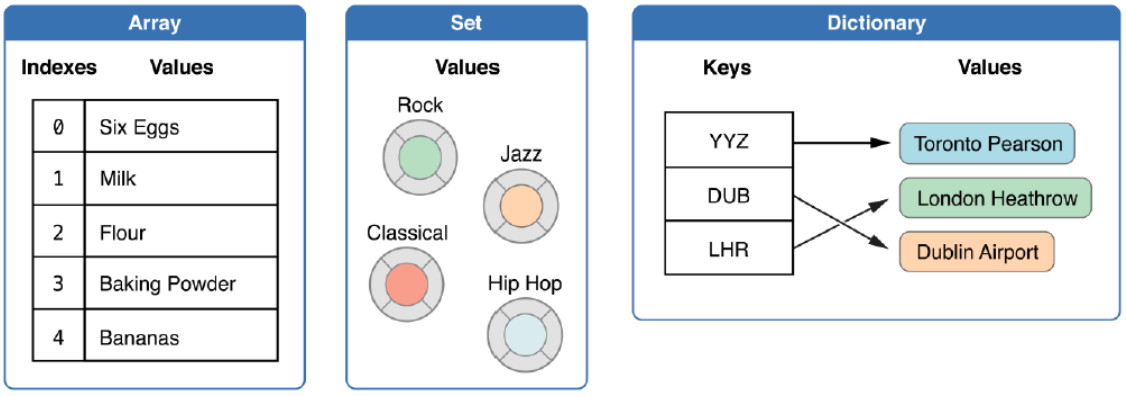
```
// Create a new Map
const map = new Map();
map.size; // 0

const map = new Map([
  ['a', 1],
  ['b', 2],
  [3, 3],
]);
map.size; // 3

// --------------------------------
// Get, Set
map.set('a', 1);
map.set('a', 2);
map.size; // 1

map.get('a'); // 2
map.get('b'); // undefined

map.delete('b'); // false
map.delete('a'); // true

map.get('a'); // undefined
map.size; // 0

// --------------------------------
// Loop through Map
const map = new Map();

map.set('a', 1);
map.set('b', 2);
map.set('c', 3);

for (const [key, value] of map) {
console.log(key, value)
}

map.forEach((value, key) => {
console.log(key, value)
})
// a 1
// b 2
// c 3

// --------------------------------
// Show city name using Map
const cityList = [
  { id: 1, name: 'TP. Hồ Chí Minh' },
  { id: 2, name: 'TP. Phan Thiết' },
]

const studentList = [
  { id: 123, name: 'Alice', cityId: 1 },
  { id: 321, name: 'Bob', cityId: 2 },
]

// Question: how to show city name for each student?
```

```
// using forEach
const cityMap = new Map();
  cityList.forEach(city => {
    cityMap.set(city.id, city);
})

// or using reduce
const cityMap = cityList.reduce((map, city) => {
  map.set(city.id, city);
  return map;
}, new Map());

cityMap.get(1).name; // TP. Hồ Chí Minh
cityMap.get(2).name; // TP. Phan Thiết
```

3 Map vs Object

	Map	Object
Accidential Keys	only contains what we add	has prototype keys, so be careful to be conflicted with your keys
Key Types	can be any type	string or symbol ONLY
Key Order	keep inserted order	better to not depend on key's order
Size	using `map.size` props	do it your self `Object.keys(obj).length`
Iteration	Yes, it support to loop	Not support, need to do it with props
Performance	Better for frequent add/remove	Not optimize for frequent add/remove
Serialization and Parsing	Do it yourself	Native support

▼ 112. JS built-in object: Set



1. Set Overview

The Set object lets you store unique values of any type, whether primitive values or object references.

You can iterate through the elements of a set in insertion order.

Some examples:

Remove duplicated numbers from [1, 2, 3, 2, 3] —> [1, 2, 3]

Remove duplicated letters from string 'aabbcc' —> 'abc'

2. Set Methods

#	Name	Desc
1	new Set()	Khởi tạo Set
2	**size** (thuộc tính, ko phải hàm)	Lấy size của set
3	add(value)	Add value mới vào set
4	has(value)	Kiểm tra set có key không?
5	delete(value)	Xoá một value
6	clear()	Xoá tất cả values
7	*keys*()	Alias của **values()**
8	*values*()	LOOP qua tất cả values
9	*entries*()	LOOP qua tất cả cặp [value, value]
10	*forEach*((value) => {})	LOOP qua tất cả values

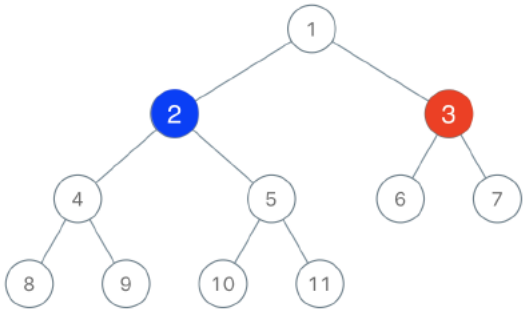
```
// Create a new Set
const set = new Set();
// Set(0) {}

const set = new Set([1, 1, 2, 2, 3, 3]);
// Set(3) { 1, 2, 3}

const set = new Set('aabbcc');
// Set(3) { 'a', 'b', 'c' }

const set = new Set(new Set([1, 2, 3]))
// Set(3) { 1, 2, 3 }

// --------------------------------
// Add, Delete
const set = new Set();
set.size; // 0

set.add(1);
set.add(2);
set.size; // 2

set.has(1); // true
set.has(3); // false

set.delete(1); // true
set.delete(3); // false as 3 is not existed

set.clear();
set.size; // 0

// --------------------------------
// Loop through Set
const numbetSet = new Set([1, 2, 3, 3]);

for (const number of numbetSet) {
console.log(number); // 1, 2, 3
}

numberSet.forEach(number => console.log(number)); // 1, 2, 3

// --------------------------------
// Remove duplicated numbers
function uniqueNumbers(numberList) {
    if (!Array.isArray(numberList) || numberList.length === 0) return [];

    const uniqueNumberSet = new Set(numberList);
    // return Array.from(uniqueNumberSet);
    return [...uniqueNumberSet];
}
```

```
console.log(uniqueNumbers([1, 1, 3, 3, 2, 2, 2]));

// --------------------------------
// Remove duplicated letters
function uniqueLetters(str) {
    if (str.length === 0) return '';

    const uniqueLetterSet = new Set(str);
    return [...uniqueLetterSet].join('');
}

console.log(uniqueLetters('abcabcabc'));

// --------------------------------
// Intersection
function getIntersectionSet(set1, set2) {
    const intersectionSet = new Set();
    for (const item of set1) {
      if (set2.has(item)) intersectionSet.add(item);
    }
    return intersectionSet;
}

const set1 = new Set([1, 2, 3]);
const set2 = new Set([2, 3, 4, 5]);
console.log(getIntersectionSet(set1, set2));
```

▼ **113. Binary Tree**

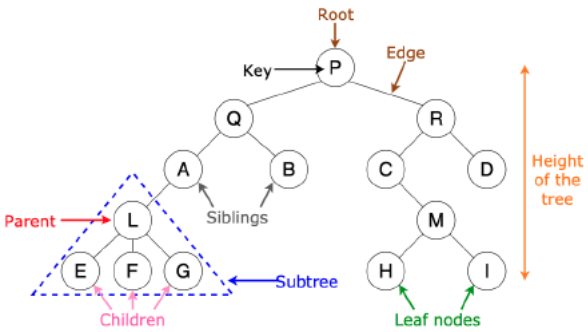
1. Characteristics

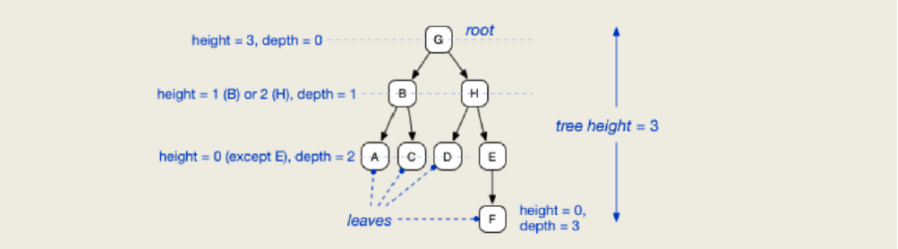
- Each node has a maximum of 2 child nodes.
- Each node contain:
  - data: number, string, object...
  - left: the left child node (possibly null)
  - right: the right child node (possibly be null)



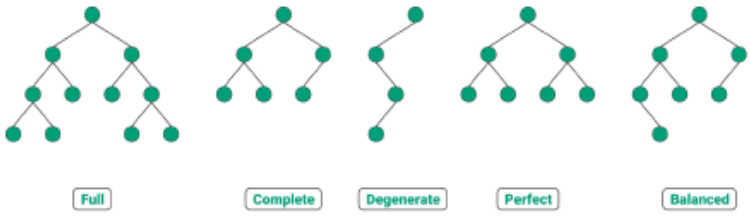
2. Terminology

#	Name	Desc
1	Root	Node trên cùng của cây
2	Parent	Tất cả node (trừ root) đều có parent node
3	Child	Node con
4	Leaf	Node ngoài cùng của cây, không có node con
5	Depth of a node	Số lượng nhánh từ root --> node
6	Height of a node	Số lượng nhánh từ node --> leaf xa nhất
7	Siblings	Các node có cùng node cha
8	Edge	Nhánh / Cạnh
9	Subtree	Cây con



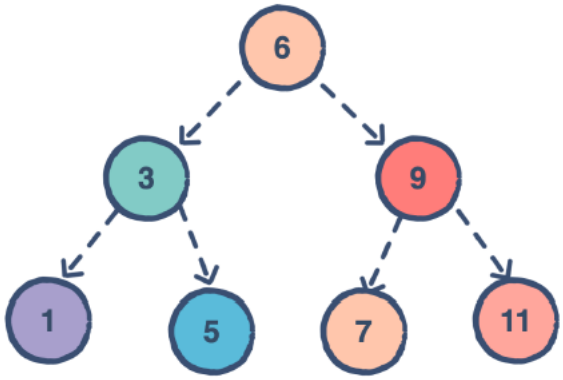


3. Binary Tree Type



4. Binary Search Tree

- The left child node is always smaller than the father node
- The right child node is always larger than the father node
- Each left and right branch is itself a Binary Search Tree.



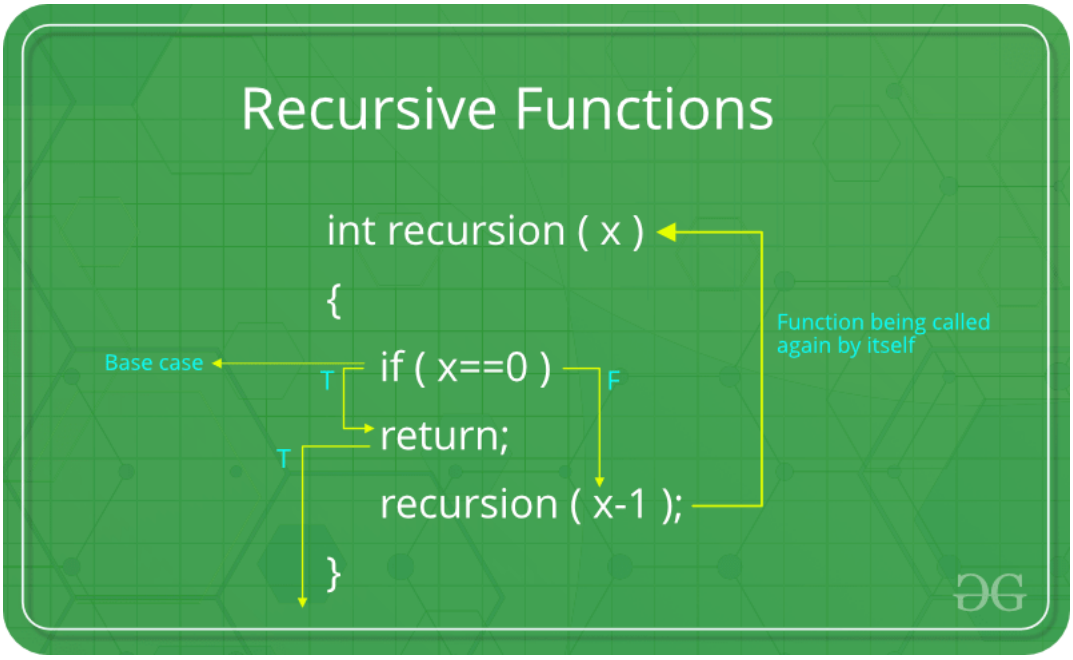
An example of a binary search tree

- Travesal (BFS, DFS)
- Search / Insert / Remove

▼ **114. Recursion**

1. What is Recursion?

> The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called as recursive function

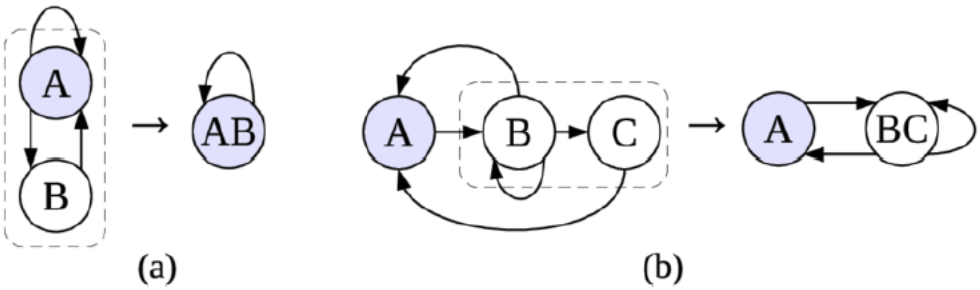


2. Notice when working with Recursion

- Must have BASE CASE (termination point), if not may have infinite loop and cause error Stack Overflow.
- The memory is incremented by the number of times a recursive function is called.
- Mobile has very limited memory space to execute any apps. If you are developing a mobile application, avoid using recursion.

3. Recursion Type

#	Name	Desc
1	Primitive recursion	Có thể chuyển đổi sang dạng loop
2	Tail recursion	Là một dạng primitive recursion mà lời gọi recursive function nó nằm ở cuối hàm.
3	Single recursion	Chỉ có một lời gọi recursive function
4	Multiple recursion	Có nhiều lời gọi recursive function
5	Mutual or Indirect recursion	Có ít nhất từ 2 functions trở lên tạo nên recursion này. `A -> B -> C -> A`
6	General recursion	Trái ngược với Primitive recursion, chỉ có thể biểu diễn dưới dạng recursive function



4. Example

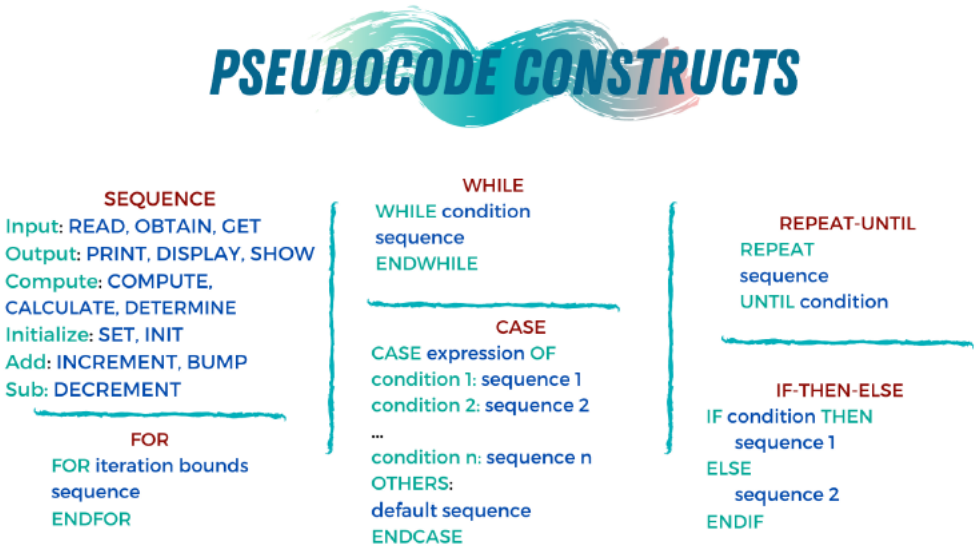
```
// Calculate: S(n) = 1 + 2 + ... + n

// using math
function calculateS(n) {
  if (n <= 0) return 0;

  return n * (n + 1) / 2;
}


// using loop
function calculateS(n) {
  if (n <= 0) return 0;

  let sum = 0;
  for (let i = 1; i <= n; i++) {
    sum += i;
  }

  return sum;
}

// using recursive function
function calculateS(n) {
  // base case
  if (n <= 0) return 0;

  // tail recursion
  return n + calculateS(n - 1);
}

calculateS(5); // 15
// S(5) = 5 + S(4)
// S(4) = 4 + S(3)
// S(3) = 3 + S(2)
// S(2) = 2 + S(1)
// S(1) = 1 + S(0)
// S(0) = 0 BASE CASE / TERMINATION
```

▼ SECTION 13: ALGORITHMS

▼ **120. Algorithm Overview**

1. Pseudo code

- What: heigh level algorithm instruction
- Why: describe an algorithm in plain language, no need to specific in any programming language
- When: research and propose solution
- How : learn some keywords to describe your algorithm




```
FUNCTION findMax
INPUT:
  numberList - an array of numbers
OUTPUT: the maximum number
---
IF (numberList is not an array or array is empty)
  RETURN undefined;

SET max = first element
FOR each number in numberList
  IF current number > max
  THEN SET max = current number

RETURN max
```

2. Evaluate the algorithm

- **Time complexity**: the time complexity of an algorithm is the amount of time taken by the algorithm to complete its process as a function of its input length, n
- **Space complexity**: the space complexity of an algorithm is the amount of space (or memory) taken by the algorithm to run as a function of its input length, n

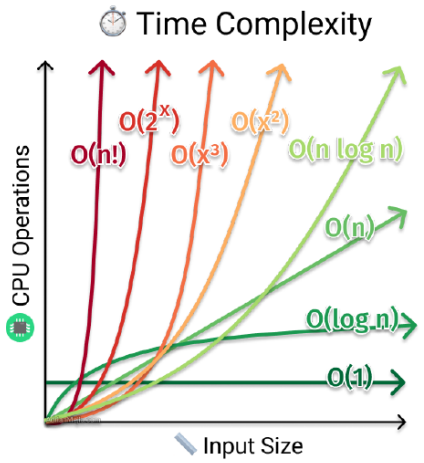
	Time Complexity	Space Complexity
What	Calc time needed	Calc memory space needed
How	Count time for all statements	Count memory for space for all vars (even input + output)
Depends on	Input data size	Mostly depends on auxiliary variables size
Important	More important for solution optimization	Less important with modern hardwares

```
function findIndex(numberList, target) {
  if (!Array.isArray(numberList) || numberList.length === 0) return -1;

  for (let i = 0; i < numberList.length; i++) {
    const number = numberList[i];
    if (number === target) return i;
  }

  return -1;
}
```

- Big O: O(n) (worst case, you need to loop to the end)
- Big Omega: O(1) (best case, you found it at the first position)

Popular Big O

#	Name	Desc
1	O(1)	fixed number of times no matter how big the input is
2	O(log(n))	reduce half on every step
3	O(n)	one loop
4	O(n^2)	two nested loop
5	O(n^3)	three nested loop

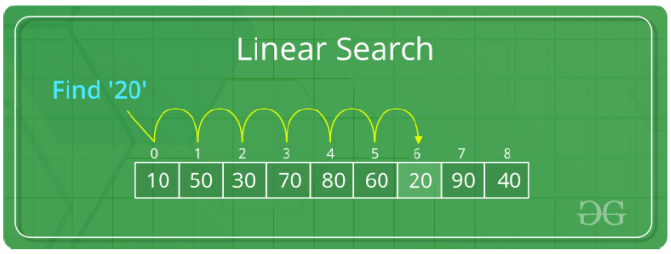


▼ **121. Search Algorithm (Visualization)**

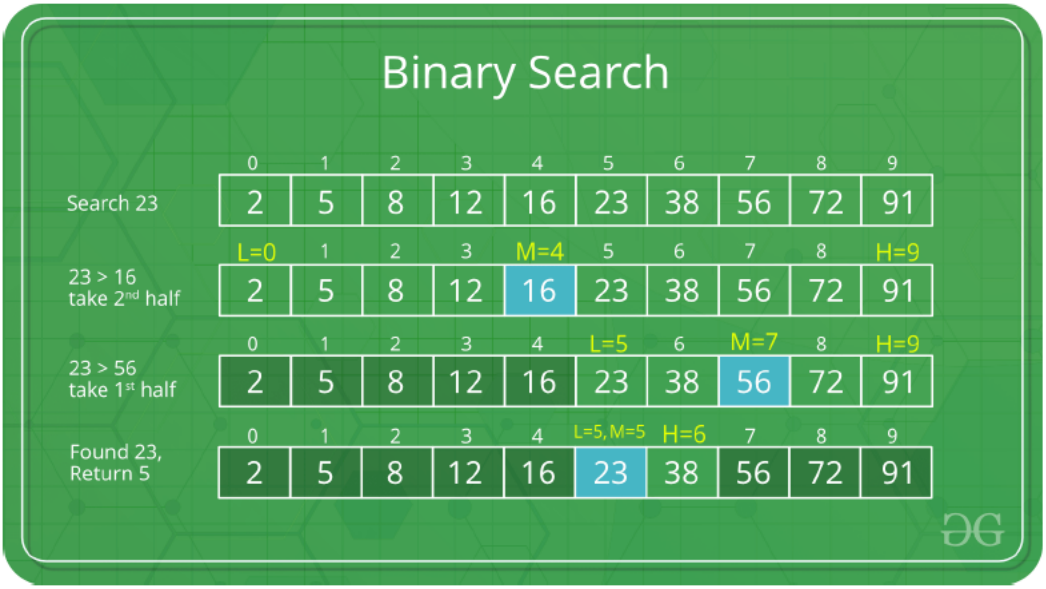
#	Linear Search	Binary Search
What	Search one by one in order	Reduce a half of search area by each step
Required sorted array	NO	**YES**
Big O	O(n)	**O(log(n))**

1. Linear Search

```
function findIndex(numberList, target) {
  if (!Array.isArray(numberList) || numberList.length === 0) return -1;

  for (let i = 0; i < numberList.length; i++) {
    const number = numberList[i];
    if (number === target) return i;
  }

  return -1;
}
```



2. Binary Search




```
//RECURSION

// assume: numberList is a sorted array
function binarySearch(numberList, target, left, right) {
  if (!Array.isArray(numberList) || numberList.length === 0) return -1;

  // base case / termination point (required for recursion)
  if (right < left) return -1;

  const mid = left + Math.trunc((right - left) / 2);
  if (numberList[mid] === target) return mid;

  // search on the right part if target is greater than mid
  if (target > numberList[mid]) {
```

```
    return binarySearch(numberList, target, mid + 1, right);
  }

  // otherwise, try to search on the left part
  return binarySearch(numberList, target, left, mid - 1);
}
```

```
// non-recursive
function binarySearch(numberList, target) {
  if (!Array.isArray(numberList) || numberList.length === 0) return -1;

  let left = 0;
  let right = numberList.length;

  while (left <= right) {
    const mid = left + Math.trunc((right - left) / 2);
    if (numberList[mid] === target) return mid;

    if (target > numberList[mid]) {
      left = mid + 1;
    } else {
      right = right - 1;
    }
  }

  return -1;
}
```

▼ **122. Bubble Sort Algorithm (Visualization)**

1. Sort Overview

Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Quicksort	O(n log(n))	O(n log(n))	O(n^2)	O(log(n))
Mergesort	O(n log(n))	O(n log(n))	O(n log(n))	O(n)
Timsort	O(n)	O(n log(n))	O(n log(n))	O(n)
Heapsort	O(n log(n))	O(n log(n))	O(n log(n))	O(1)
Bubble Sort	O(n)	O(n^2)	O(n^2)	O(1)
Insertion Sort	O(n)	O(n^2)	O(n^2)	O(1)
Selection Sort	O(n^2)	O(n^2)	O(n^2)	O(1)
Shell Sort	O(n)	O((nlog(n))^2)	O((nlog(n))^2)	O(1)
Bucket Sort	O(n+k)	O(n+k)	O(n^2)	O(n)
Radix Sort	O(nk)	O(nk)	O(nk)	O(n+k)

2. Bubble Sort

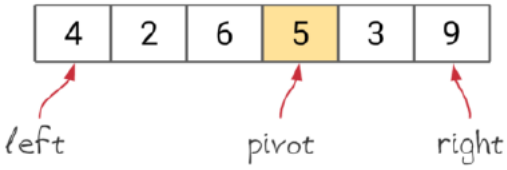
```
function bubbleSort(numberList) {
  if (!Array.isArray(numberList) || numberList.length === 0)
    return numberList;

  for (let i = numberList.length - 1; i > 0; i--) {
    for (let j = 0; j < i; j++) {
      if (numberList[j] > numberList[j + 1]) {
```

```
        // swap if left item is greater than right item
        const temp = numberList[j];
        numberList[j] = numberList[j + 1];
        numberList[j + 1] = temp;
      }
    }
  }

  return numberList;
}

console.log(bubbleSort([5, 3, 2, 1, 6, 9]));
// [ 1, 2, 3, 5, 6, 9 ]
```

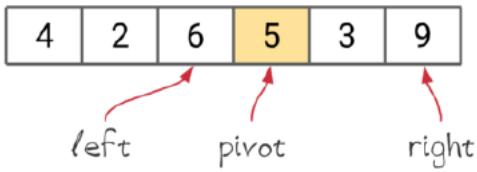
▼ **122. Quick Sort Algorithm (Visualization)**

1. Partition

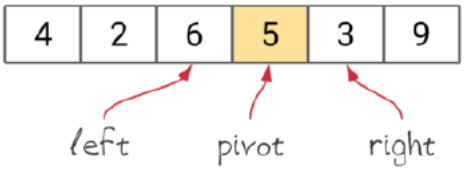
Step 1: Determine a pivot



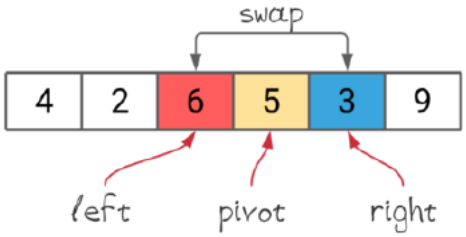
Step 2: Shift the left pointer to find a value greater than the pivot



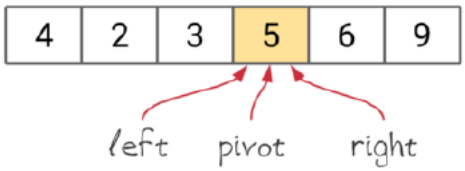
Step 3: Shift the right pointer to find a value less than the pivot



Step 4: Swap values at the pointers



Step 5: Repeat the about steps until two pointers both meet or cross



2. Quick Sort

```
function partition(numberList, left, right) {
  //   console.log('partition', numberList.slice(left, right + 1));
  const mid = left + Math.trunc((right - left) / 2);

  const pivot = numberList[mid];
  let i = left;
  let j = right;

  while (i <= j) {
    // find the first item that greater or equal than pivot
    while (numberList[i] < pivot) i++;

    // find the first item that smaller or equal than pivot
    while (numberList[j] > pivot) j--;

    if (i < j) {
      const temp = numberList[i];
      numberList[i] = numberList[j];
      numberList[j] = temp;
    }

    // if i <= j we will increase,
    // case i > j not increase
    if (i <= j) {
      // MUST HAVE, INF loop
      i++;
      j--;
    }
  }

  return i;
}

function quickSort(numberList, left, right) {
  //   console.log('quick sort', numberList.slice(left, right + 1));
  // base case
  if (left >= right) return numberList;

  const pivotPosition = partition(numberList, left, right);

  quickSort(numberList, left, pivotPosition - 1);
  quickSort(numberList, pivotPosition, right);

  return numberList;
}
```

```
console.log(quickSort([4, 2, 3], 0, 2));
// 4 2 3
// i j
// 2 4 3
// j i = 1
// 4 3
// i j
// 3 4
// j i


console.log(quickSort([1], 0, 0));
console.log(quickSort([1, 2], 0, 1));
console.log(quickSort([1, 2, 1], 0, 2));
console.log(quickSort([1, 2, 1, 3], 0, 3));
console.log(quickSort([4, 1, 2, 5], 0, 3));
console.log(quickSort([4, 2, 6, 5, 3, 9], 0, 5));
```

▼ SECTION 14: ES6 AND SOME CONCEPTS IN JAVASCRIPT

▼ **126. Strict mode**

1. Strict mode

- Introduced in ECMAScript 5
- A flag to tell that you'd like to use modern code (from ES5 upward)

2. Advantages

- Eliminates some JavaScript silent errors by changing them to throw errors.
- Fixes mistakes that make it difficult for JavaScript engines to perform optimizations: strict mode code can sometimes be made to run faster than identical code that's not strict mode.
- Prohibits some syntax likely to be defined in future versions of ECMAScript.

3. Invoking strict mode

Invoke for the whole script by adding 'use strict' on top of the file

```
// Whole-script strict mode syntax
'use strict';

const message = "Hi! I'm a strict mode script!";

function sayHello() {}
```

Invoke for current function by adding 'use strict' on top of the function

```
function strict() {
  // Function-level strict mode syntax
  'use strict';

  function nested() {
  return 'And so am I!';
  }

  return "Hi! I'm a strict mode function! " + nested();
  }

  function notStrict() {
    return "I'm not strict.";
}
```

for **classes**, **es6 modules**, it auto turn of strict mode by default

4. Strict mode help prevent some errors

Impossible to accidentally create global variables

```
// in non-strict mode
channelName = 'Easy Frontend'; // accidentally create global variables
console.log(channelName); // 'Easy Frontend'
console.log(window.channelName); // 'Easy Frontend' --> browser only
console.log(global.channelName); // 'Easy Frontend' --> nodejs server only

'use strict';
channelName = 'Easy Frontend';
// ReferenceError: channelName is not defined
```

Turn silently fail to throw an exception

```
// in non-strict mode
undefined = 1; // no error

'use strict'
undefined = 1;
// TypeError: Cannot assign to read only property 'undefined' of object
```

Throw error when you attempt to delete undeletable properties

```
// in non-strict mode
delete Object.prototype; // no effect and no error too 🤣

'use strict';
delete Object.prototype;
// TypeError: Cannot delete property 'prototype' of function Object() {
[native code] }
```

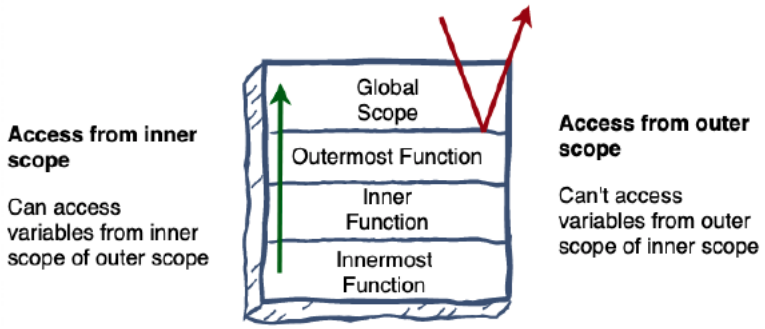
Octal literals are not allowed in strict mode

```
// in non-strict mode
console.log(015 + 20); // 33
// because 015 is understand in octal
// 015 = 5 * 8^0 + 1 * 8^1 = 13

'use strict';
console.log(015 + 20);
// SyntaxError: Octal literals are not allowed in strict mode.
// correct way: use 0o (zero and small 'o' letter)
console.log(0o15 + 20);
```

▼ **127. Scope**

1. Scope Overview

- **What**: the limits in which a variable exists
- **Why**: to understand when and where we can access to a variable
- **When**: declare a variable
- **Types of Scope**: block scope, function scope, lexical scope, global scope



2. Block Scope

- Block scope = limit in the pair of curly brackets (braces) {}
- **const/let** has block scope (ES6)

```
{
  const name = 'Easy Frontend';
}
console.log(name); // runtime error: name is not defined

// nested block scope
{
  {
    const name = 'Easy Frontend';
  }
console.log(name); // runtime error: name is not defined
}

if (n > 10) {
  const name = 'Easy Frontend';
}
console.log(name); // runtime error: name is not defined

for (let i = 0; i < 10; i++) {
  const name = 'Easy Frontend';
}
console.log(name); // runtime error: name is not defined

// how about this case?
for (let i = 0; i < 10; i++) {
  if (i % 5 === 0) {
    const isDivisibleToFive = true;
}
  console.log(isDivisibleToFive); // ???
}
console.log(isDivisibleToFive); // ???
```

3. Function Scope

- function scope = limit into current function only.

```
function sayHello() {
  const name = 'Easy Frontend';
  console.log(name); // 'Easy Frontend'

  function print() {
    const language = 'en';
    console.log(language); // 'en'
  }

  print();
  console.log(language);
  // ReferenceError: language is not defined
}
```

```
sayHello();

console.log(name);
// ReferenceError: name is not defined
```

4. Lexical Scope

- Lexical scope = ability for a function scope to access variables from the parent scope.
- lexical scope ~ where is this scope defined?
- lexical scope ~ the definition area of an expression
- lexical scope has another name static scope
- lexical scope is the place in which the item got created

```
const name = 'Easy Frontend';

function sayHello() {
  const language = 'en';
  console.log(name);
  // the lexical scope of name is global scope

  function printLanguage() {
    console.log(language);
    // the lexical scope of language is a function scope (sayHello)
  }
}
```

5. Global Scope

- global scope can be accessed anywhere
- global scope = declare variable not inside any function

```
let count = 1;
let channelName = 'Easy Frontend'
```

#	Brower	NodeJS	Worker
global object	window	global	WorkerGlobalScope / self

- In JavaScript, there's always a global object defined
- The global object's interface depends on the execution context in which the script is running
- Global object is an object that always exists in the global scope.
- The global **globalThis** property contains the global **this** value, which is asking to the global object.

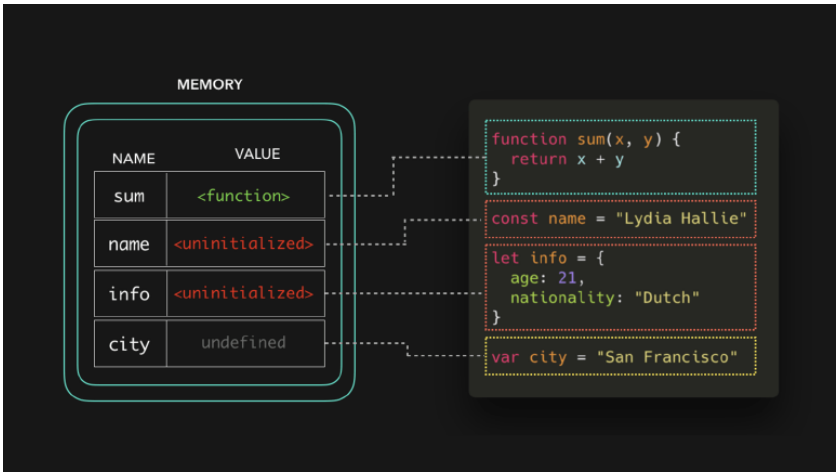
```
var getGlobal = function () {
  if (typeof self !== 'undefined') { return self; }
  if (typeof window !== 'undefined') { return window; }
  if (typeof global !== 'undefined') { return global; }
  throw new Error('unable to locate global object');
};
```

6. Scope Chain

A scope chain refers to the unique spaces that exist from the scope where a variable got called to the global scope.

```
// First fullName variable defined in the global scope:
const fullName = "Easy";

// Nested functions containing two more fullName variables:
function profile() {
  const fullName = "Frontend";

  function sayName() {
    const fullName = "Boooom";

    function writeName() {
      return fullName;
    }
    return writeName();
  }

  return sayName();
}
```

▼ **128. Hoisting**



1. What is Hoisting?

> JavaScript Hoisting refers to the process whereby the compiler allocates memory for **variable** and **function declarations** prior to execution of the code.

#	Hoisting	Example
var	**YES**	`var count = 1;`
const / let	NO	`let name = 'Easy';`
function declaration	**YES**	`function sayHello() {}`
function expression	NO	`const sum = function() {}`

💡 **IMPORTANT**: Only declarations are hoisted, not initializations.

```
console.log(num); // Returns 'undefined' from hoisted var declaration
var num; // Declaration
num = 6; // Initialization

// no var declaration
console.log(num); // ReferenceError: num is not defined
num = 6; // Initialization
```

2. Function Declaration Hoisting

- Able to use function before the declaration.

```
sum(1, 2); // also works
function sum(a, b) {
  return a + b;
}

sum(1, 2); // works
```

BE CAREFUL with function expression, the var is hoisted with undefined value, not the function.

```
sum(1, 2); // TypeError: sum is not a function
// because sum is var
// and get hoisted with undefined value

var sum = function (a, b) {
    return a + b;
};

sum(1, 2); // works
```

```
var tip = 100;
(function () {
  console.log("I have $" + husband());

  function wife() {
    return tip * 2;
  }

  function husband() {
    return wife() / 2;
  }

  var tip = 10;
})();
```

3. Var, Const, Let

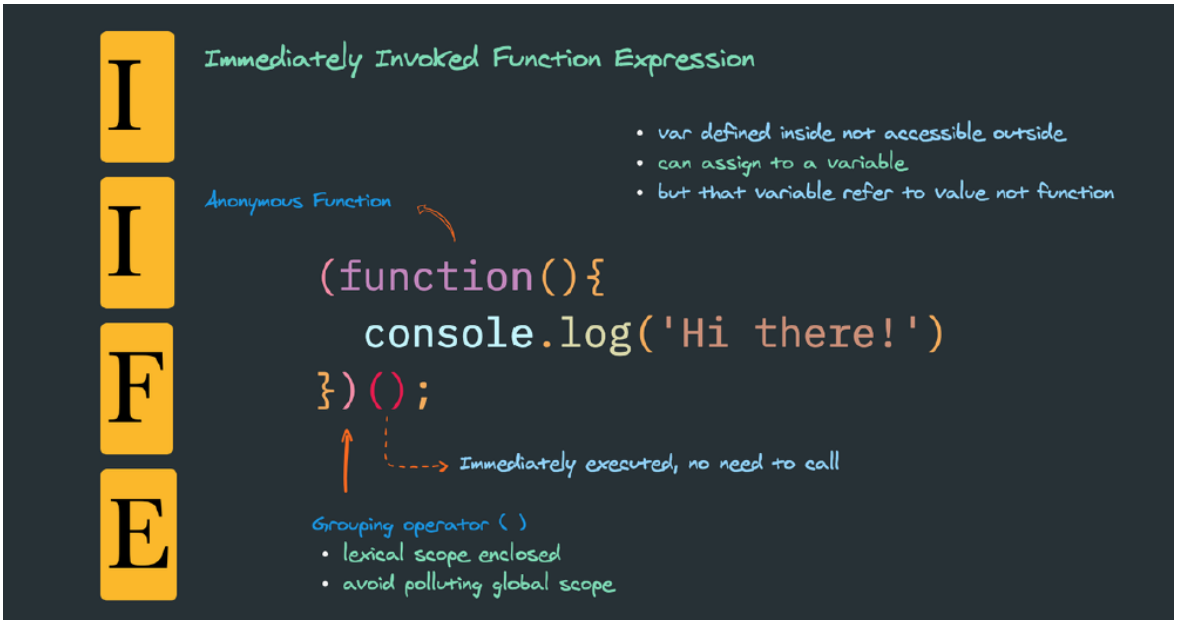
	var	const	let
initializing	optional	required	optional
re-declare (same scope)	yes	no	no
can be re-assigned	yes	no	yes
scope	function or global	block or global	block or global
hoisting	yes	no	no
when to use	avoid using it	constant, unchanged vars	changable vars
become a member of window in global	yes	no	no

```
var channelName = '';
var channelName = 'Easy Frontend'; // it works =))

const channelName = '';
const channelName = 'Easy Frontend';
// Uncaught SyntaxError: Identifier 'channelName' has already been declared

let channelName = '';
let channelName = 'Easy Frontend';
// Uncaught SyntaxError: Identifier 'channelName' has already been declared

// function scope vs block scope
function sayHello() {
  {
    const language = 'en'; // block scope
    var message = 'hello'; // function scope
  }

console.log(language); // ReferenceError: language is not defined
console.log(message); // 'hello'
}
```

4. Temporal Dead Zone

💡 **Const/let hoisting**?
⇒ Yes, but with Temporal Dead Zone (cannot be used before the declaration line).
⇒ The underlying of const/let is also hoisting, BUT it adds a concept of Temporal Dead Zone to check and detect the use of const/let variable before declaring, it will give an error.

▼ 130. IIFE - Immediately Invoke Function Expression



1. Problem

Sometimes, there are functions created to be used only once.

```
function main() {
  console.log('Hien Tran');
}

main();

function App() {
  useEffect(() => {
    function fetchData() {
    // fetching data
    }

    fetchData();
  }, [])

return null;
}
```

2. Introduce IIFE

- **What**: An IIFE (Immediately Invoked Function Expression) is a JavaScript function that runs as soon as it is defined
- **When**: function has only one call
- **Why**: shorter, no need to name it.

```
// syntax
(function () {
  console.log('IIFE');
})();

// IIFE with arrow function
(() => {})();

(() => {
  console.log('IIFE');
})();

// IIFE with params
((a, b) => {
  console.log(a + b);
```

```
})(5, 10);

// IIFE with async arrow function
(async () => {})();

(async () => {
  await fetchData();
})();
```

3. IIFE in Real Life

```
// BEFORE
function main() {
  console.log('Hien Tran');
}

main();

// AFTER
(() => {
  console.log('Hien Tran');
})();


// BEFORE
function App() {
  useEffect(() => {
    async function fetchData() {
      // fetching data
    }

    fetchData();
  }, [])

  return null;
}

// AFTER
function App() {
  useEffect(() => {
    (async () => {
      // fetch data
    })();
  }, []);

  return null;
}
```

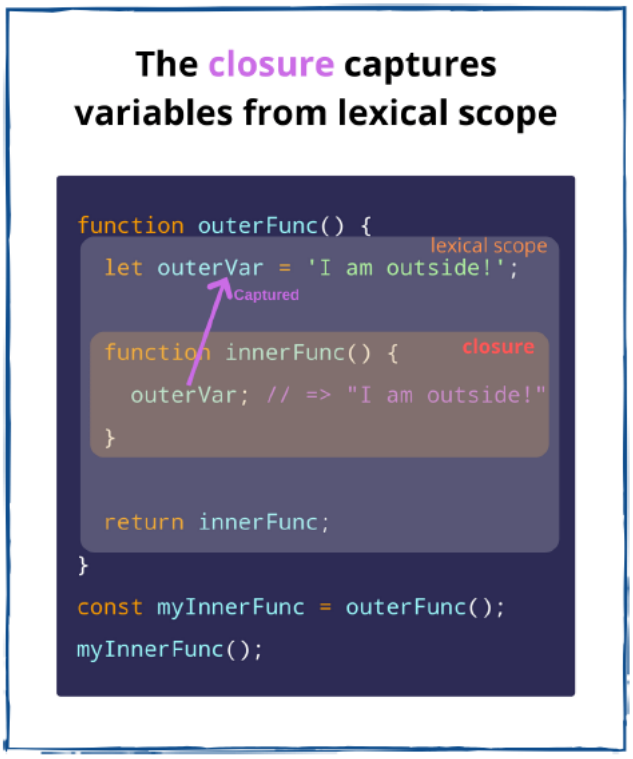
4. Prefer block scope instead of IIFE

```
(() => {
  var name = 'Easy Frontend';
})();

console.log(name); // ReferenceError: name is not defined


// Prefer way
{
  let name = 'Easy Frontend';
}

console.log(name); // ReferenceError: name is not defined
```

5. IIFE from Google

```
▼<script nonce>

    (function(d,e,v,s,i,t,E){d['GoogleDevelopersObject']=i;
      t=e.createElement(v);t.async=1;t.src=s;E=e.getElementsByTagName(v)[0];
      E.parentNode.insertBefore(t,E);})(window, document, 'script',
      'https://www.gstatic.com/devrel-devsite/prod/ve25c3e040096d9860a1b86503ff88692bbcb336f820486
  '[1,"en",null,"/js/devsite_app_module.js","https://www.gstatic.com/devrel-devsite/prod/ve25c3e04
  devsite/prod/ve25c3e040096d9860a1b86503ff88692bbcb336f8204869702074147185c6987/developers","http
  ["/_pwa/developers/manifest.json","https://www.gstatic.com/devrel-devsite/prod/ve25c3e040096d986
  devsite/prod/ve25c3e040096d9860a1b86503ff88692bbcb336f8204869702074147185c6987/developers/images
  family=Google+Sans:400,500|Roboto:400,400italic,500,500italic,700,700italic|Roboto+Mono:400,500,
  [1,6,8,12,14,17,21,25,40,50,63,70,75,76,80,87,91,92,93,97,98,100,101,102,103,104,105,107,108,111
  jjEJBzmIyKR4F-3XITp8yM9T1gEEI8","AIzaSyB6xiKGDR5O3Ak2okS4rLkauxGUG7XP0hg"]')

</script>
```


▼ 131. Closures



1. Definition

A closure is a **function** having **access to the parent scope**, **even after** the **parent function has closed**.

```
function init() {
  var name = 'Mozilla'; // name is a local variable created by init

  function displayName() { // the inner function, a closure
    alert(name); // use variable declared in the parent function
  }
  displayName();
}
init();


function makeFunc() {
```

```
  var name = 'Mozilla';
  function displayName() {
    alert(name);
  }
  return displayName;
}

var myFunc = makeFunc();
myFunc();
```

2. Counter Example

```
function createCounter(initValue = 0) {
  let value = initValue; // private variable

  function increase() {
    value++;
  }

  function decrease() {
    value--;
  }

  function getValue() {
    return value;
  }

  return {
    getValue,
    increase,
    decrease,
  };
}


const counter = createCounter();
counter.getValue(); // 0
counter.increase();
counter.increase();

counter.getValue(); // 2
counter.decrease();
counter.getValue(); // 1
console.log(counter.value); // undefined
```

▼ 132. Function

1. Default parameter value

```
// move default param to the right most
function sum(a, b) {}
function sum(a, b = 10) {}
function sum(a = 5, b = 10) {}

function sum(a = 5, b = 10) {
  return a + b;
}

sum(); // 15
sum(10); // 20
sum(10, 20); // 30

sum(undefined, undefined); // 15
sum(undefined, null); // 5 as null is converted to 0
```

2. Rest Parameter

```
// ES5
function sum() {
  let sum = 0;
```

```
  for (let i = 0; i < arguments.length; i++) {
    sum += arguments[i];
}

  return sum;
}

console.log(sum(1)); // 1
console.log(sum(1, 2)); // 3
console.log(sum(1, 2, 3)); // 6
```

💡 NOTE: **arguments** object is an Array-like object, not Array

```
// convert arguments to Array
function sum() {
  return [...arguments].reduce((total, number) => total + number);
}

console.log(sum(1, 2, 3));


// ES6 - Prefer this way instead of arguments
function sum(...numberList) {
  return numberList.reduce((total, number) => total + number);
}

console.log(sum(1, 2, 3));
```

3. Spread Operator

```
function sum(...numberList) {
  return numberList.reduce((total, number) => total + number);
}

console.log(sum(1, 2, 3)); // 6

const numberList = [1, 2, 3];
console.log(sum(...numberList)); // 6
```

4. Arrow Function

```
function sum() {}; // function declaration
const sum = function() {}; // function expression
const sum = () => {}; // arrow function


const sum = (a, b) => {
  return a + b;
}

// shorthand
const sum = (a, b) => a + b;
```

5. Constructor Function

```
function Student(id, name) {
  this.id = id;
  this.name = name;

  this.sayHello = function () {
    console.log('My name is', this.name);
  };
}
const alice = new Student(1, 'Alice');
alice.sayHello(); // My name is Alice
```

```
const bob = new Student(2, 'Bob');
const bob = new Student(2, 'Bob');
```

6. Curry Function

Curry function / Higher Order Function (HOF)

```
sum(1)(2); // 3

function sum(x) {
  return function (y) {
    return x + y;
  };
}


// generate increase id
function createIdGenerator(startId = 1) {
  let id = startId;

  return function() {
    return id++;
  }
}

const getNextId = createIdGenerator(10);
getNextId(); // 10
getNextId(); // 11
getNextId(); // 12
```

▼ **133. ES6 - Enhance Object Props**

1. Property Shorthand

- Use property shorthand when key and variable name are the same.
- If no override needed, then please keep the shorthand props on top.

```
const id = 1;
const name = 'Easy Frontend';
const age = 18;

const student = {
  id: id, // key and variable name are the same
  name: name,
  age: age,
  isHero: false,
}

// shorthand version
const student = {
  id,
  name,
  age,
  isHero: false,
}
```

2. Computed Property Name

- Used for key having spaces.
- Used for key that need to build from other variables.

```
const key = 'Power';

const student = {
  id: 1,
  name: 'Easy Frontend',
```

```
  'hero type': 'iron man', // key with spaces

  [key]: 50,

  [`get${key}`]: function() {
    return 100;
  }
}

student.id; // 1
student.Power; // 50

student.hero type; // syntax error
student['hero type']; // 'iron man'

student.Power; // 50
student[key]; // 50

student.getPower(); // 100
```

3. Method Properties

```
const student = {
  sayHello: function() {
    console.log('Easy Frontend')
}, // ES5

  getPower() {
    return 100;
    }, // ES6
}
```

4. Desctructuring

```
// object destructuring
const student = {
  id: 1,
  name: 'Easy Frontend',
}

const { id, name } = student;


// array destructuring
const numberList = [5, 10, 15];
const [a, b] = numberList;
// a = 5
// b = 10


// swap
let x = 10;
let y = 20;

[y, x] = [x, y]; // swap

console.log(x); // 20
console.log(y); // 10



// rename prop
// destructuring value
const student = {
  id: 1,
  name: 'Easy Frontend',
}

const { id: studentId, name, age = 18 } = student;
console.log(studentId); // 1
console.log(age); // 18

console.log(id); // ReferenceError: id is not defined
```

5. Object.assign()

```
const a = { id: 1 }
const b = { name: 'Easy' }
const c = Object.assign(a, b);

console.log(a); // { id: 1, name: 'Easy' }
console.log(c); // { id: 1, name: 'Easy' }

console.log(a === c); // true (same reference)


// clone object
const student = {
  id: 1,
  name: 'Easy Frontend',
}


// spread operator
const alice = {
  ...student,
  name: 'Alice'
};


// object.assign
const bob = Object.assign({}, student, { name: 'Bob' })
```

▼ **134. This in JavaScript**

1. Overview

- this = current execution context
- this can be different in strict and non-strict mode
- this has different value depending on where it is used:

In	this refers to
alone / outside function	global object
normal function in strict mode	undefined
normal function	depends on how the func is called, default to global object
arrow function	lexical this / this from outer normal function
a method	owner object
an event	an element that received the event

2. this in Global Context


```
// browser
console.log(this); // window object
console.log(this === window); // true

this.name = 'Easy Frontend';
console.log(window.name); // 'Easy Frontend'

// nodejs
console.log(this); // global object
console.log(this === global); // true

this.name = 'Easy Frontend';
console.log(global.name); // 'Easy Frontend'
```

3. this in Function Context

```
// non-strict mode
function sayHello() {
  console.log(this); // window or global
}

// anonymous function
[1, 2, 3].forEach(function(number) {
  console.log(this); // window or global
})


'use strict'
function sayHello() {
  console.log(this); // undefined
}

// anonymous function
[1, 2, 3].forEach(function(number) {
  console.log(this); // undefined
})
```

4. this in Arrow Function

- arrow function doesn't have it own this.
- this from outer normal function

```
const sayHello = () => {
  console.log(this); // window or global
}

'use strict'
function sayHello() {
```

```
  console.log(this); // undefined

  const getLanguage = () => {
    console.log(this); // same as this from outer normal function
  }

  getLanguage();
}
```

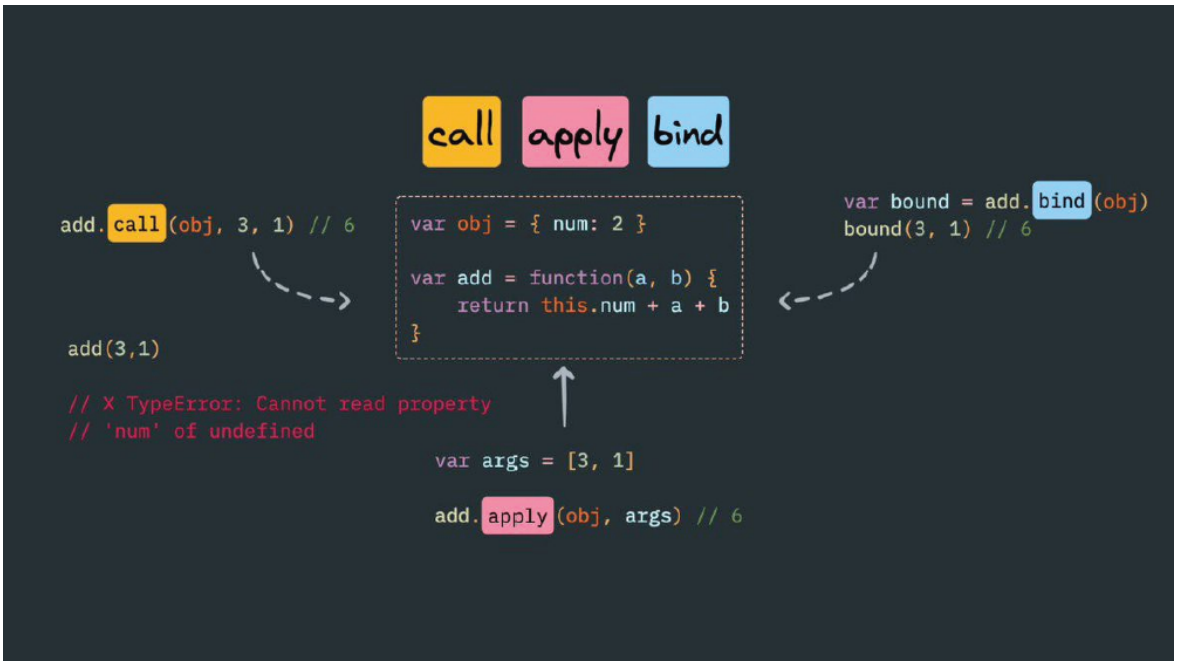
5. this in a Method

```
const student = {
  name: 'Bob',
  sayHello: function() {
    console.log('My name is', this.name);
  },
}

student.sayHello(); // 'My name is Bob'


const student = {
  name: 'Bob',
  // ES6 property methods
  sayHello() {
    console.log('My name is', this.name);
  }
}
student.sayHello(); // 'My name is Bob'


'use strict'
// avoid using arrow function in object methods
const student = {
  name: 'Bob',

  // arrow function
  sayHello: () => {
    console.log('My name is', this.name);
  }
}

student.sayHello(); // 'My name is undefined'
```

6. this in an Event (later)

7. this in Class Context (later)

▼ **135. bind, call, apply**



> Src: https://javascript.plainenglish.io/quick-guide-to-call-apply-and-bind-methods-in-javascript-5c00cd856cfa

#	bind	call	apply
bind this context	yes	yes	yes
return	new function	result of func call	result of func call
time of binding	now	now	now
time of exec	future	now	now
params		arguments	an array of params

💡 **NOTE**: Only works with regular function, NOT arrow function.

```
'use strict';

function sayHello(a, b) {
  console.log(this.name, a + b);
}

const student = {
  name: 'Hien Tran',
```

```
};

const studentSayHello = sayHello.bind(student);
studentSayHello(5, 10);

sayHello.call(student, 10, 20);
sayHello.apply(student, [1, 2]);
```

▼ **136. Iteration Protocols**

1. Iteration Protocols

- Iteration protocols **aren't** built-ins or syntax, but protocols.
- These protocols **can be implemented by any object** by following some conventions.
- There are two protocols: **iterable protocol** and **iterator protocol**

2. The Iterable Protocol

- The iterable protocol allows JavaScript objects to define or customize their iteration behavior.
- Built-in iterables: String, Array, TypedArray, Map, and Set.
- To be iterable, an object must implement the **@@iterator** method, it returns an object, confirming to the **iterator protocol**.

```
const numberList = [5, 10, 15];

for (const number of numberList) {
  console.log(number);
}

// Example of iterable
const iterable = {
  [Symbol.iterator]() {
    return {}; // this object should be an iterator
  }
}
```

3. The Iterator Protocol

- The iterator protocol defines a standard way to produce a sequence of values (either finite or infinite), and potentially a return value when all values have been generated.
- To become an iterator, an object must implement **next()** method, **return { value, done }**

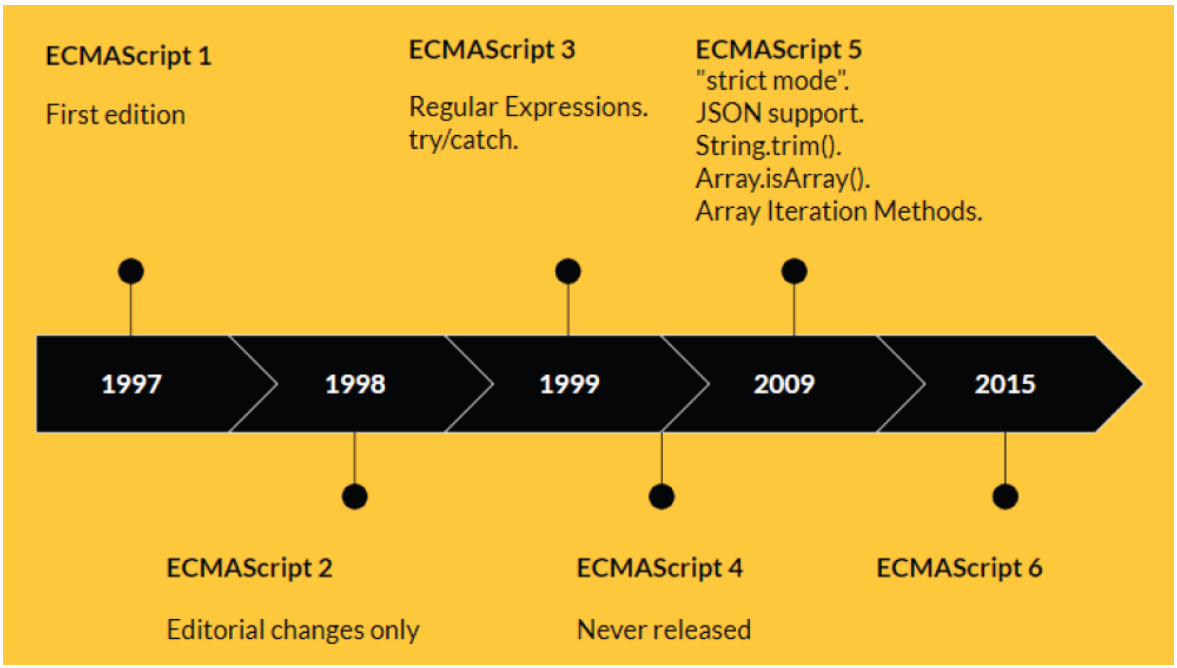
```
const iterator = {
  next() {
    return { value: 1, done: true },
  }
}
```

▼ SECTION 15: ECMASCRIPT FEATURES

Year	Features
2016	`arr.includes()` expo operator
2017	async func shared memory and atomics string padding `obj.values()` `obj.entries()` `obj.getOwnPropertyDescriptors()` trailing commas
2018	rest/spread for object asynchronus iteration `promise.property.finally()` regular expression improvements

Year	Features
2019	`arr.protorype.{flat, flatMap}` `obj.fromEntries()` `string,prototype.{trimStart, trimEnd}` `symbol.prototype.description` optional catch binding `arr.prototype,sort()` is guaranteed to be stable revised `func.prototype.toString()`
2020	bigInt dynamic import nullish coalescing optional chaining `promise.allSettled` `string.prototype.matchAll` `globalThis` module namespace exports well defined for-in order `import.meta`
2021	logical assignment operator numeric separator `string.prototype.replaceAll()` `promise.any` weakrefs and finalizers

▼ **138. ECMAScript Overview**



> Src: https://www.goconqr.com/c/64835/course_modules/108563-es6-next-generation-javascript

1. Overview

Ecma International is a nonprofit standards organization for information and communication systems

2. Standards

#	Name	Desc
1	ECMA-262	ECMAScript Language Specification (based on JavaScript)
2	ECMA-402	ECMAScript Internationalization API Specification
3	ECMA-404	JSON

3. Committees

#	Name	Desc
1	**TC39 - Technical Committee 39**	responsible for **ECMAScript**
2	TC45 - Technical Committee 45	responsible for Office Open XML
3	TC53 - Technical Committee 53	responsible for ECMAScript Modules for Embedded Systems

4. TC39

#	Name	Note
1	Responsibilies	read
2	Meeting	every two months, mostly in U.S
3	Release	yearly in June
4	Version	ES6 ... ES2020, ES2021, ...

5. TC39 Process

Stage	Name	What
Stage 0	strawman	represents an initial idea for addition or change to the specification
Stage 1	proposal	is a formal proposal describing a problem and suggesting a proper solution
Stage 2	draft	is an initial draft of the proposal specification
Stage 3	candidate	represents the draft when it's almost final but ready for last feedback
Stage 4	finished	completely ready and included within the next edition

6. Proposal List

> Reference: https://github.com/tc39/proposals

▼ **139. ECMAScript 2016**

Key	Value
Name	ECMAScript 2016
Short name	ES2016 / ES7
Was finalized in	June 2016

1. Feature List

#	Name	Desc
1	Array.prototype.includes()	new method for Array to check if an item is existed
2	Exponentiation Operator	similar to **Math.pow()**

2. `Array.prototype.includes()`

```
[1, 2].includes(1); // true
[1, 2].includes(2); // true
[1, 2].includes(3); // false


['easy', 'frontend'].includes('easy'); // true
['easy', 'frontend'].includes('frontend'); // true
['easy', 'frontend'].includes('easy frontend'); // false
```

3. Exponentiation Operator

The ** operator is standardized across many languages including Python, Ruby, MATLAB, Lua, Perl and many others.

```
Math.pow(2, 3); // 8
console.log(2 ** 3); // 8
```

> Reference: https://flaviocopes.com/es2016/

▼ 140. ECMAScript 2017

Key	Value
Name	ECMAScript 2017
Short name	ES2017
Was finalized in	June 2017

1. Feature List

#	Feature	Desc
1	MAJOR Async functions	higher level abstraction over promises
2	MAJOR Shared memory and atomics	shared memory between multiple workers
3	String padding	add characters to reach a specific length. source
4	Object.values()	get an array of object values
5	Object.entries()	get an array of pair [key, value]
6	Object.getOwnPropertyDescriptors()	get object property descriptor
7	Trailing commas	in function parameter lists and calls

2. Major Async Functions

The async and await keywords enable asynchronous, promise-based behavior to be written in a cleaner style.

```
function main() {
  getAllStudents()
    .then(response => {
      const { data, pagination } = response;
      console.log(data, pagination);
    })
    .catch(error => {
      console.log(error);
    })
}



async function main() {
  try {
    const response = await getAllStudents();
    console.log(data, pagination);
  } catch (error) {
    console.log(error);
  }
}
```

3. Major Shared Memory and Atomics

Since ES2017, you can **create a shared memory array** between **web workers and their creator**, using a **SharedArrayBuffer**.

> Reference: https://github.com/tc39/ecmascript_sharedmem/blob/master/TUTORIAL.md

4. String Padding

- pad = add character until it reach a specific length.
- default pad character is **space**

```
'1234'.padStart(8); // ' 1234'
'1234'.padStart(8, '*'); // '****1234'
```

```
'123456'.padStart(8, '*'); // '**123456'

'1234'.padEnd(8, '*'); // '1234****'
'12'.padEnd(8, '*'); // '12******'
```

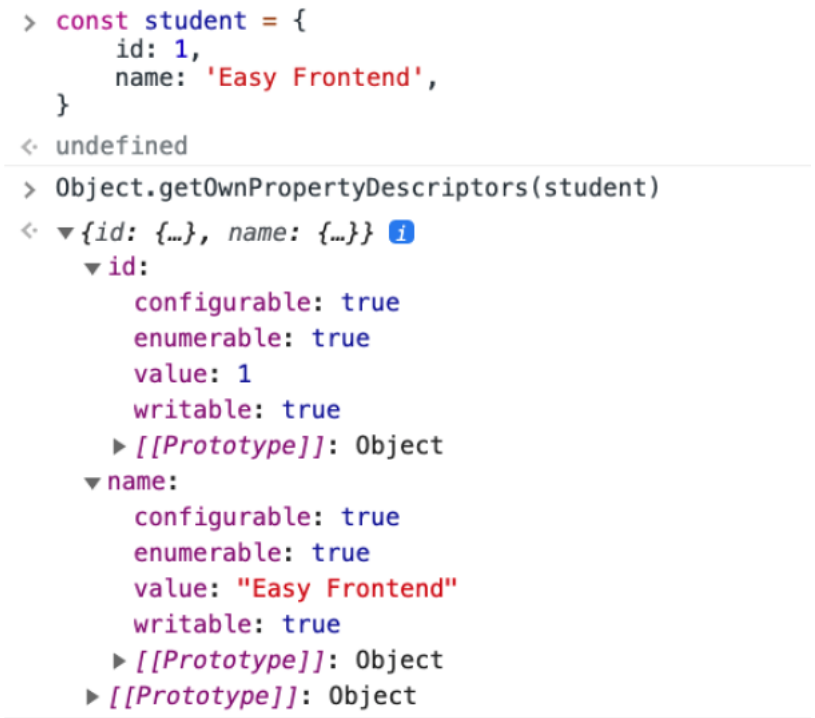
5. `Object.values()`

```
Object.values({
  id: 1,
  name: 'Easy Frontend',
  age: 18,
});

// [1, 'Easy Frontend', '18']
```

6. `Object.entries()`

```
Object.values({
  id: 1,
  name: 'Easy Frontend',
  age: 18,
});

// [
// ['id', 1],
// ['name', 'Easy Frontend'],
// ['age', 18],
// ]
```

7. `Object.getOwnPropertyDescriptors()`



8. Trailing commas

Trailing commas in Array

```
[1, 2, 3,].length; // 3
[1, 2, 3,,].length; // 4

const numberList = [1, 2, 3,];
console.log(numberList); // [1, 2, 3]
```

Trailing commas in object (from ECMAScript 5)

```
const student = {
  id: 1,
  name: 'Easy Frontend',
  age: 18, // <-- trailing comma
}
```

Trailing commas in functions (from ECMAScript 2017)

```
// function declaration
function createStudent({
  id,
  name,
  age,
  hobbies,
  isHero, // trailing comma
}) {
  console.log(id, name, age, hobbies, isHero);
}

// function call
createStudent({
  id: 1,
  name: 'Easy Frontend',
  age: 18,
  hobbies: ['music', 'coding'],
  isHero: false, // trailing comma
})
```

Trailing commas when using git

> Src: https://pawelgrzybek.com/trailing-comma-in-ecmascript2017-function-parameter-list/

▼ 141. ECMAScript 2018

Key	Value
Name	ECMAScript 2018
Short name	ES2018
Was finalized in	June 2018

1. Feature List

#	Name	Desc
1	MAJOR Rest/Spread properties for objects	similar to rest/spread element for array (ES6)
2	MAJOR Asynchronus iteration	for-await-of
3	Promise.prototype.finally()	always run no matter failed or success
4	Regular Expression improvements	leave it for now, will get back to this in regex section

2. Rest/Spread Properties for Objects

ES6 / Rest element

```
const [first, second, ...rest] = [1, 2, 3, 4, 5]
// first = 1
// second = 2
// rest = [3, 4, 5]
```

ES6 / Spread element

```
const numberList1 = [1, 2, 3];
const numberList2 = [4, 5, 6];
const mergedList = [...numberList1, ...numberList2, 7, 8, 9];
// [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

ES2018 introduces the same but for objects.

ES2018 / Rest properties

```
const student = {
  id: 1,
  name: 'Easy Frontend',
  isHero: false,
}

const { id, ...rest } = student;
// id = 1
// rest = { name: 'Easy Frontend', isHero: false }
```

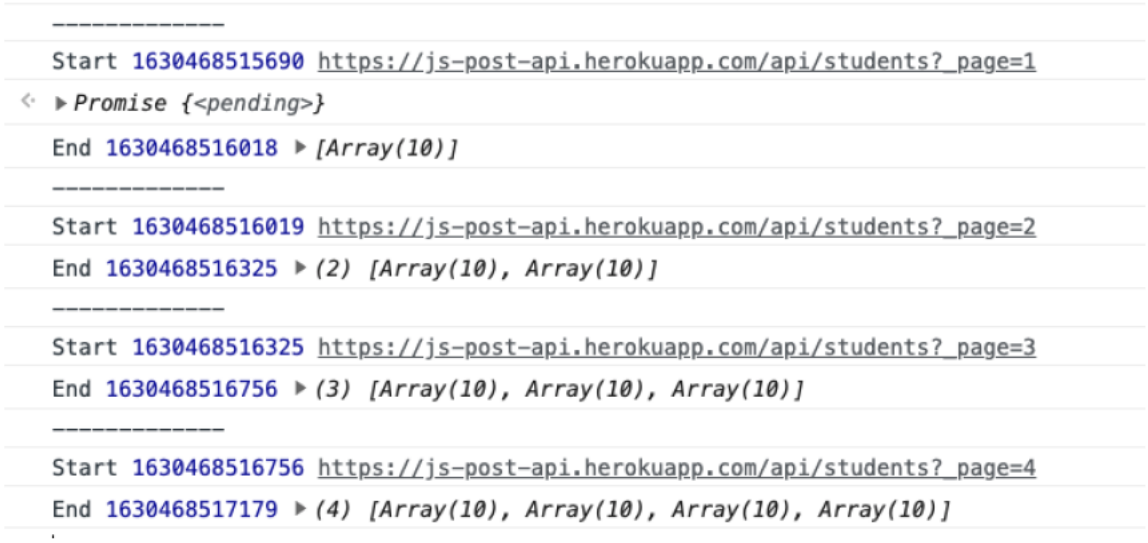
ES2018 / Spread properties

```
const alice = {
  id: 1,
  name: 'Alice',
}

const bob = {
  ...alice,
  name: 'Bob',
}
```

3. Asynchronous iteration

Syntax: for-await-of

```
async function fetchData() {
  const studentList = [];
  const pageList = [1, 2, 3, 4];

  for await (const page of pageList) {
    const url = `https://js-post-api.herokuapp.com/api/students?
    _page=${page}`;

    console.log('-------------');
    console.log('Start', new Date().getTime(), url);

    const response = await fetch(url);
    const responseJSON = await response.json();

    studentList.push(responseJSON.data);
    console.log('End', new Date().getTime(), studentList)
  }
}
fetchData();
```



4. `Promise.prototype.finally()`

```
fetch('file.json')
  .then(data => data.json())
  .catch(error => console.error(error))
  .finally(() => console.log('finished'))
```

5. Regular Expression Improvements

> Reference: https://flaviocopes.com/es2018/

▼ **142. ECMAScript 2019**

Key	Value
Name	ECMAScript 2019
Short name	ES2019
Was finalized in	June 2019

1. Feature List

#	Name	Desc
1	MAJOR Array.prototype.{flat, flatMap}	flatten array
2	MAJOR Object.fromEntries	create object from entries
3	String.prototype.{trimStart, trimEnd}	trim whilte spaces
4	Symbol.prototype.description	add desc to a Symbol
5	Optional catch binding	a way to omit the error in catch
6	Array.prototype.sort() is guaranteed to be stable	keep same order for equal items
7	Revised Function.prototype.toString()	keep original function declaration including comments, spaces

> Src: https://exploringjs.com/es2018-es2019/ch_overview.html

2. `Array.prototype.{flat, flatMap}`

arr.flat(depth) - default depth is 1

```
[1, 2, [3, 4]].flat(0); // [1, 2, [3, 4]]
[1, 2, [3, 4]].flat(1); // [1, 2, 3, 4]

[1, 2, [[3], 4]].flat(1); // [1, 2, [3], 4]
[1, 2, [[3], 4]].flat(2); // [1, 2, 3, 4]

[1, 2, , 4, 5].flat(); // [1, 2, 4, 5] as empty slot is removed
```

arr.flatMap(mapFn) - similar to the call map() first, then flat(1). But slightly more efficient

```
[5, 20]
  .map(x => [x, x * 2]) // [ [5, 10], [20, 40] ]
  .flat(1); // [5, 10, 20, 40]


[5, 20].flatMap(x => [x, x * 2]); // [5, 10, 20, 40]
```

3. `Object.fromEntries`

Create new object from entries.

```
const student = {
  id: 1,
  name: 'Easy Frontend',
}

// ES2017
const entries = Object.entries(student);
// [
//     ['id', 1],
//     ['name', 'Easy Frontend'],
// ]

// ES2019
Object.fromEntries(entries);
// {
//    id: 1,
  //    name: 'Easy Frontend',
// }
```

4. `String.prototype.{trimStart, trimEnd}`

Easy to remove redundant whitespaces at the beginning / end of a string.

```
// ES5
' Easy Frontend '.trim(); // 'Easy Frontend'

// ES2019
'Easy Frontend'.trimStart(); // 'Easy Frontend'
' Easy Frontend '.trimStart(); // 'Easy Frontend '

'Easy Frontend'.trimEnd(); // 'Easy Frontend'
' Easy Frontend '.trimEnd(); // ' Easy Frontend'
```

5. `Symbol.prototype.description`

```
const sym = Symbol('description goes here');
sym.description; // 'description goes here'
sym.toString(); // 'Symbol(desc)'
```

6. Optional Catch Binding

Able to omit the error in catch clause if we don't need it.

```
function parseStudentFromJSON(str) {
  try {
    return JSON.parse(str);
  } catch (error) {
    // in case of failed to parse, simply return undefined
    return undefined;
    // error is unused 😱
  }
}



function parseStudentFromJSON(str) {
  try {
    return JSON.parse(str);
  } catch {
    // in case of failed to parse, simply return undefined
    return undefined;
  }
}
```

7. `Array.prototype.sort()` is Guaranteed to be Stable

Guaranteed to keep the same order for equal items.

```
[
  { id: 1, name: 'Alice' },
  { id: 3, name: 'John' },
  { id: 2, name: 'Bob' },
  { id: 4, name: 'Alice' },
].sort((s1, s2) => {
  if (s1.name > s2.name) return 1;
  if (s1.name < s2.name) return -1;

  return 0;
});

// output
// [
//   { id: 1, name: 'Alice' },
//   { id: 4, name: 'Alice' },
//   { id: 2, name: 'Bob' },
//   { id: 3, name: 'John' },
// ]
```

8. Revised `Function.prototype.toString()`

ES2019 introduced a change to the return value to **avoid stripping comments** and **other characters like whitespace**, exactly representing the function as it was defined.

```
function /* this is bar */ bar() {}

// BEFORE
bar.toString() //'function bar() {}

// ES2019
bar.toString() // 'function /* this is bar */ bar () {}'


// for built-in function
Math.trunc.toString();
// function trunc() { [native code] }
```

> Reference: https://flaviocopes.com/es2019/

▼ 143. ECMAScript 2020

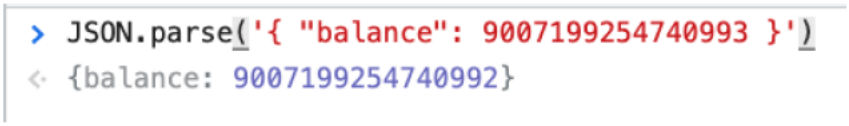
Key	Value
Name	ECMAScript 2020
Short name	ES2020
Was finalized in	June 2020

1. Feature List

#	Name	Desc
1	BigInt	New primitive type. Big integer can represent number larger than 2^53 - 1
2	Dynamic Import	"function-like" import() module loading syntactic
3	**Nullish Coalescing**	`x ?? y` means return y if x is nullish
4	**Optional Chaining**	access deep property without checking each ref in the chain is valid
5	Promise.allSettled	wait until all promises finished, either fulfilled or rejected
6	String.prototype.matchAll	leave it for now, get back later in regular expression section
7	**globalThis**	Get the global object of current environment.
8	**Module Namespace Exports**	export * as something from 'module'
9	Well defined for-in order	standardlize the enumeration order of for-in
10	import.meta	access meta information about the module

2. BigInt

```
Number.MAX_SAFE_INTEGER; // 2^53 - 1 = 9007199254740991
Number.MAX_SAFE_INTEGER + 2; // 9007199254740992 ??? incorrect calculation
const balance = BigInt('9007199254740991');
balance + 2n; // 9007199254740993n
```




```
JSON.parse('{ "balance": 9007199254740993 }'); // { balance: 9007199254740992 }

// correct way to handle it
const json = '{ "balance": "9007199254740993" }';
const { balance } = JSON.parse(json);
BigInt(balance); // 9007199254740993n
```

> Reference: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/BigInt

3. Dynamic Import

"function-like" import() module loading syntactic. Source

```
// math.js
const add = (num1, num2) => num1 + num2;
export { add };

// index.js
const doMath = async (num1, num2) => {
if (num1 && num2) {
const math = await import('./math.js');
console.log(math.add(5, 10));
};
};
doMath(4, 2);
```

> Reference: https://www.digitalocean.com/community/tutorials/js-es2020

4. Nullish Coalescing

```
const value = x ?? y;
// - return y if x is nullish (null or undefined)
// - otherwise return x


null ?? 'Easy Frontend'; // 'Easy Frontend'
undefined ?? 'Easy Frontend'; // 'Easy Frontend'

'' ?? 'Easy Frontend'; // ''
0 ?? 'Easy Frontend'; // 0
Number.NaN ?? 'Easy Frontend'; // NaN
false ?? 'Easy Frontend'; // false

null || 'Easy Frontend'; // 'Easy Frontend'
undefined || 'Easy Frontend'; // 'Easy Frontend'

'' || 'Easy Frontend'; // 'Easy Frontend'
0 || 'Easy Frontend'; // 'Easy Frontend'
Number.NaN || 'Easy Frontend'; // 'Easy Frontend'
false || 'Easy Frontend'; // 'Easy Frontend'
```

> Reference: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Nullish_coalescing_operator

5. Optional Chaining

- it returns undefined if some ref in the chain is nullish (null or undefined)

```
obj?.prop // optional static property access
obj?.[expr] // optional dynamic property access

arr?.[index] // optional array item access
func?.(args) // optional function or method call
```

```
const student = {
  id: 1,
  address: {
    city: 'HCMC',
  },
  hobbies: ['music'],
  sayHello() {
    console.log('Hello');
  },
}

student.address.city; // 'HCMC'
student.hobbies[0]; // 'music'
student.sayHello(); // 'Hello'

// but what if you're trying to access non-existing props in the chain
student.profile.name;
// TypeError: Cannot read property 'name' of undefined

student.colors[0];
// TypeError: Cannot read property '0' of undefined

student.learnJavascript();
// TypeError: student.learnJavascript is not a function


if (student.profile) {
console.log(student.profile.name);
}

// Optional chaining
student.profile?.name; // undefined
student.colors?.[0]; // undefined
student.learnJavascript?.(); // undefined
```

6. `Promise.allSettled`

Able to omit the error in catch clause if we don't need it.

```
const p1 = new Promise((res, rej) => setTimeout(res, 1000));
const p2 = new Promise((res, rej) => setTimeout(rej, 1000));
Promise.allSettled([p1, p2]).then(data => console.log(data));

// Output
[
  Object { status: "fulfilled", value: undefined},
  Object { status: "rejected", reason: undefined}
]
```

> Reference: https://www.digitalocean.com/community/tutorials/js-es2020

7. `String.prototype.matchAll`

> Reference: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/matchAll

8. `globalThis`

Get the global object of current environment.

Environment	globalThis value
browser	window
nodejs	global
web workers	self

```
// before ES2020
var getGlobal = function () {
if (typeof self !== 'undefined') { return self; }
if (typeof window !== 'undefined') { return window; }
if (typeof global !== 'undefined') { return global; }
throw new Error('unable to locate global object');
};

var globals = getGlobal();

if (typeof globals.setTimeout !== 'function') {
  // no setTimeout in this environment!
}


// when working with server side rendering,
// you may see this code block to check whether it's client or server
if (typeof window !== 'undefined') {
  // it's on client side
} else {
  // it's on server side
}
```

9. Module Namespace Exports

```
// before ES2020
import * as utils from './utils.mjs'
export { utils }

// ES2020
export * as utils from './utils.mjs'
```

10. Well Defined for-in Order

The ECMA specification did not specify in which order for (x in y) should run. Even though browsers implemented a consistent order on their own before now, this has been officially standardized in ES2020.

> Reference: https://www.freecodecamp.org/news/javascript-new-features-es2020/

11. `import.meta`

You can access meta information about the module using the import.meta object:

```
console.log(import.meta); // { url: "file:///home/user/module.js" }
```

> Reference: https://www.freecodecamp.org/news/javascript-new-features-es2020/

▼ 143. ECMAScript 2021

Key	Value
Name	ECMAScript 2021
Short name	ES2021
Was finalized in	June 2021

1. Feature List

#	Name	Desc
1	Logical Assignment Operators	??=, ||=, &&=
2	Numeric separator	1_500_000
3	String.prototype.replaceAll()	replace all matches with new value
4	Promise.any	resolve the value of **the first fulfilled promise**
5	WeakRefs and Finalizers	read more

2. Logical Assignment Operators

#	Name	Desc
1	Logical OR assignment ||=	x ||= y means only assigns x = y if x is **falsy**.
2	Logical AND assignment &&=	x &&= y means only assigns x = y if x is **truthy**.
3	Logical Nullish assignment ??=	x??= y means only assigns x = y if x is **nullish**

```
//"Or Or Equals"
x ||= y;
x || (x = y);

// "And And Equals"
x &&= y;
```

```
x && (x = y);

// "QQ Equals"
x ??= y;
x ?? (x = y);
```

Logical OR assignment examples.

```
const a = { duration: 50, title: '' };

a.duration ||= 10;
console.log(a.duration);
// expected output: 50

a.title ||= 'title is empty.';
console.log(a.title);
// expected output: "title is empty"
```

Logical AND assignment examples

```
let a = 1;
let b = 0;

a &&= 2;
console.log(a);
// expected output: 2

b &&= 2;
console.log(b);
// expected output: 0
```

Logical Nullish assignment examples

```
const a = { duration: 50 };

a.duration ??= 10;
console.log(a.duration);
// expected output: 50

a.speed ??= 25;
console.log(a.speed);
// expected output: 25
```

3. Numeric separator

```
// before
const count = 1000000;

// ES2021 - more readable
const count = 1_000_000;

// don't overuse it =)))
1_00; // 100
10_0; // 100
1_5_0; // 150;
```

4. `String.prototype.replaceAll()`

```
// Before: need to use regex with global flag
const queryString = 'q=query+string+parameters';
```

```
const withSpaces = queryString.replace(/\+/g, ' ');
// 'q=query string parameters'

// ES2021
const queryString = 'q=query+string+parameters';
// const withSpaces = queryString.split('+').join(' ');
queryString.replaceAll('+', ' ');
// 'q=query string parameters'
```

Comparison between String.prototype.replace() and String.prototype.replaceAll()

searchValue is	replace()	replaceAll()
a string	replace first match	replace all matches
a non-global regex	replace first match	throw error
a global regex	same	same

> Reference: https://github.com/tc39/proposal-string-replaceall

5. `Promise.any()`

```
Promise.any([
  fetch('https://v8.dev/').then(() => 'home'),
  fetch('https://v8.dev/blog').then(() => 'blog'),
  fetch('https://v8.dev/docs').then(() => 'docs')
]).then((first) => {
  // Any of the promises was fulfilled.
  console.log(first);
  // → 'home'
}).catch((error) => {
  // All of the promises were rejected.
  console.log(error);
});
```

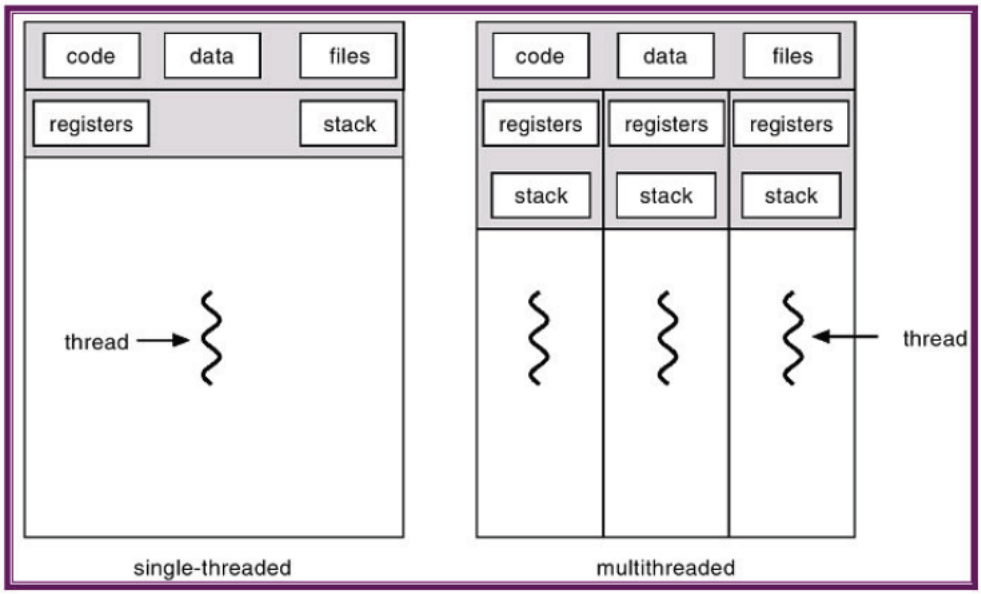
- Promise.race = wait for the first item either fulfilled or rejected.
- Promise.any = wait for the first fulfilled item

6. WeakRefs and Finalizers

▼ SECTION 16: ASYNCHRONOUS PROGRAMMING

▼ **146. Aysnchronous Overview**

1. Single Thread and Multiple Thread



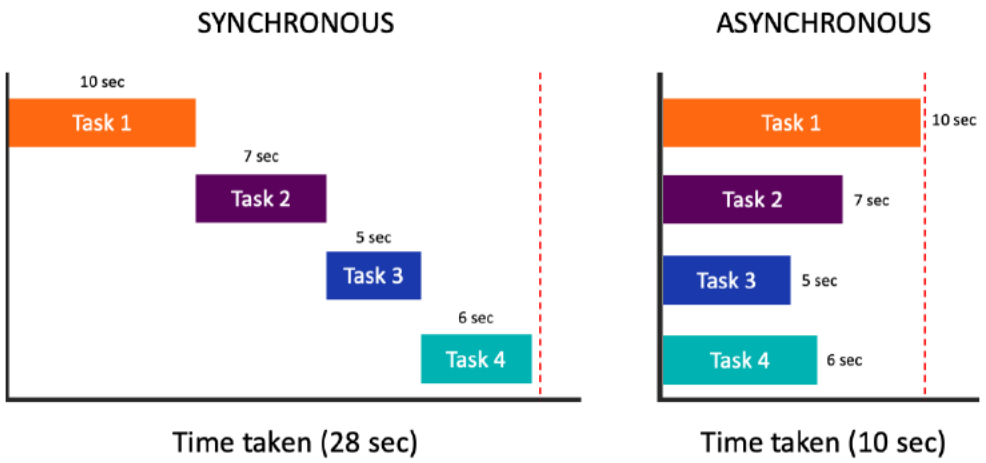
> Reference: https://dev.to/hrishi2710/threading-in-operating-system-3gjb

	Single Thread	Multi Thread
How many task performed at a time	One	Many
Resource sharing	No	Yes

⇒ **Javascript** is a **single thread** programming language.

2. Synchronous and Asynchronous

- Sync = synchronous
- Async = asynchronous
- Why async? --> avoid blocking main thread while processing



> Src: https://scoutapm.com/blog/async-javascript

> Src; https://david-gilbertson.medium.com/should-you-should-be-using-web-workers-hint-probably-not-9b6d26dc8c6a

	sync	async
Executed in sequence	YES	NO
How many tasks are running at a time	One	Many (concurrency model)
Blocking	YES	NO
Which functions?	Math.trunc(), [].slice(), console.log(), ...	setTimeout, setInterval, Promise, async/await

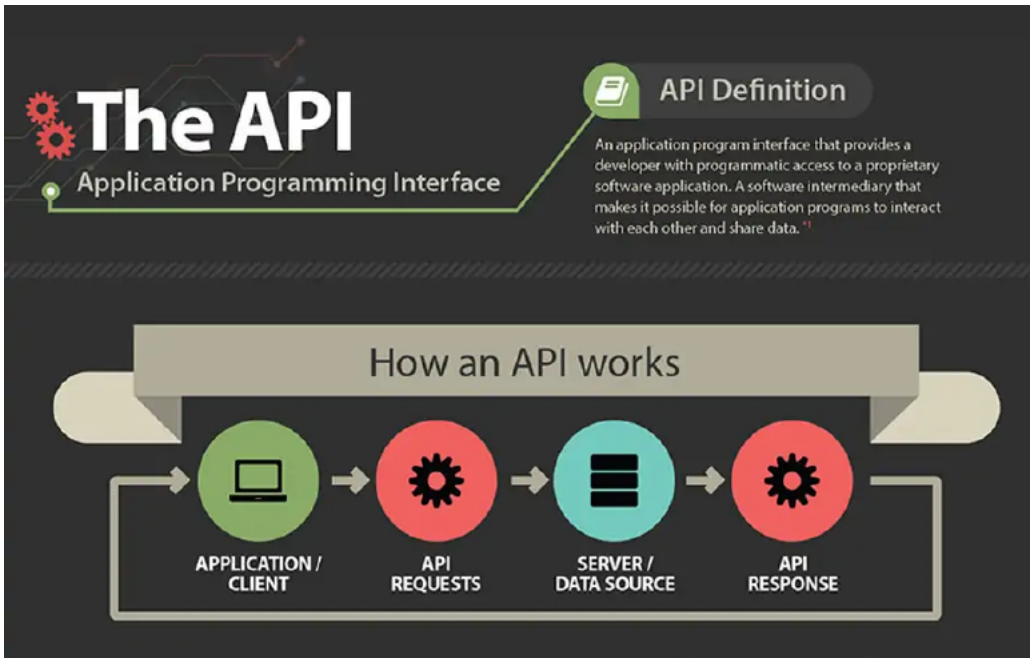
```
const sum = 1 + 2;
console.log(sum);
```

3. Blocking on Brower

#	Name	Desc
1	window.**alert**(message)	Show an alert with message
2	window.**confirm**(message)	Return true/false based on user click OK/Cancel
3	window.**prompt**(message, default)	Return value that user enter into the box

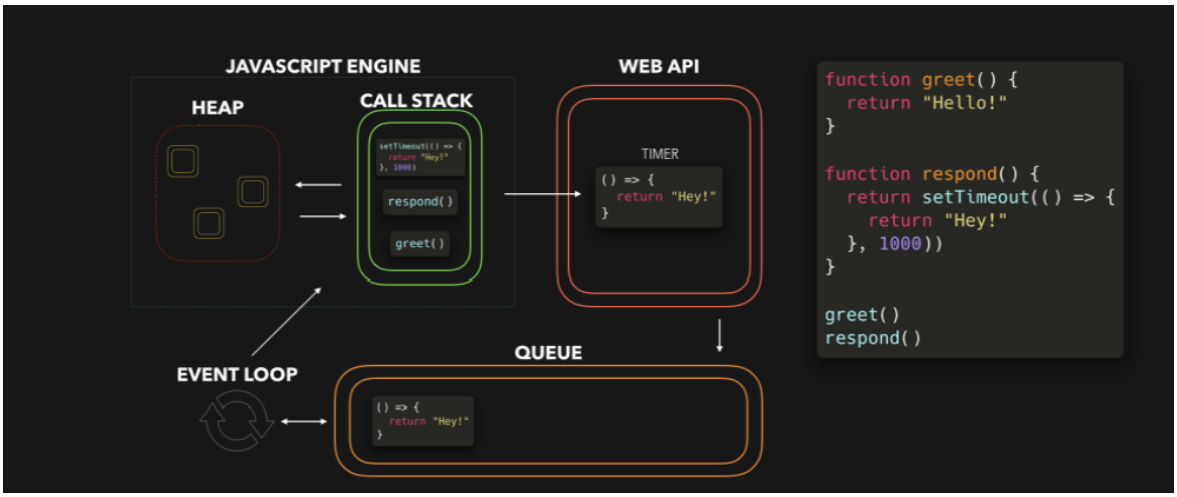
```
window.alert('Easy Frontend');
console.log('Tada!!!'); // won't exec until user click OK on alert dialog
```

4. API



> Src: https://www.smartfile.com/blog/the-api-infographic/

▼ **148. setTimeout(callback, timeout)**



> Reference: https://dev.to/lydiahallie/javascript-visualized-event-loop-3dif

💡 **Web APIs**: https://developer.mozilla.org/en-US/docs/Web/API

1. setTimeout overview

- **What**: a global method to set a timer, then exec a function when the timer is expired.
- **When**: you want to run a function after a period of time.
- **Where**: working on both browser + nodejs

```
const timeoutId = setTimeout(callback, timeout);
// callback is the function will be executed once the timer is expired
// timeout is the time in milliseconds, default to 0. 1 second = 1000 milliseconds

// timeoutId is a positive integer, used to identify a timeout
// and clear it by using clearTimeout(timeoutId)
```

```
function sayHello() {
  console.log('Hi');
}

// say hello after 1 second
setTimeout(sayHello, 1000);


// say hello after 1 second
setTimeout(() => {
  console.log('Hi');
}, 1000);


// say hello as soon as you can (not immediately)
setTimeout(() => {
  console.log('Hi');
}, 0);

setTimeout(() => {
  console.log('Hi');
};

console.log('log 1');
```

```
setTimeout(() => { console.log('log 2') };
console.log('log 3');

// log 1
// log 3
// log 2 🤣


// Real case: Redirect to another page after 3 seconds
setTimeout(() => {
  window.location.href = 'https://google.com';
}, 3000);
```

2. clearTimeout

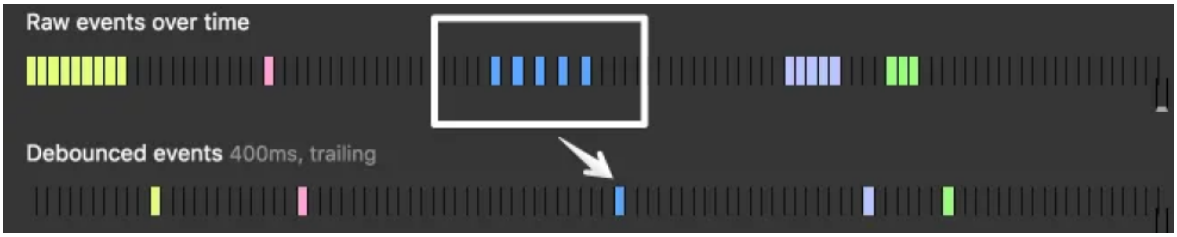
```
const timeoutId = setTimeout(() => {
  console.log('Tada!!!')
}, 5000);

clearTimeout(timeoutId); // cancel the timeout
// should be called before the timeout expired
```

3. Late timeout

- The timeout can also fire later than expected if the page (or the OS/browser) is busy with other tasks.

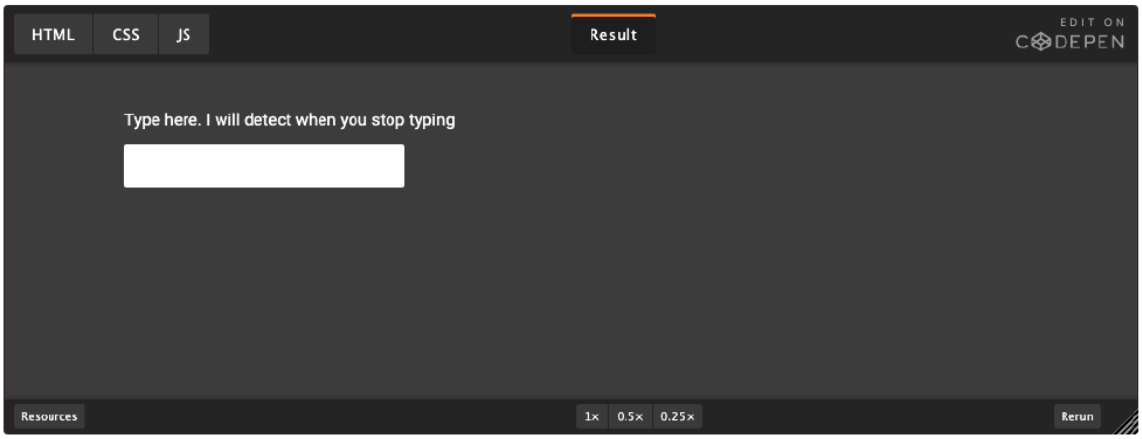
```
setTimeout(() => {
  console.log('Tada!!!')
});

let count = 1;
for (let i = 0; i < 1e9; i++) {
  count++;
}

console.log('done');
```

▼ 149. Debounce Technique



> Reference: https://css-tricks.com/debouncing-throttling-explained-examples/

1. Debounce demo



> Visualize: https://redd.one/blog/debounce-vs-throttle/

2. How it works

- debounce = exec function after a period of time receiving no trigger.

```
function debounce(callback, wait) {}

function log() {}

const debounceLog = debounce(log, 500);
debounceLog();
debounceLog();
debounceLog();
```

3. Simple Implementation

```
function debounce(callback, wait) {
  let timeoutId;

  return function () {
    if (timeoutId) {
      clearTimeout(timeoutId);
    }

      timeoutId = setTimeout(callback, wait);
  };
}

function log() {
  console.log('tada');
}

log();
log();
log();
log();
log();
// log tada 5 times

const debounceLog = debounce(log, 500);
debounceLog();
debounceLog();
debounceLog();
debounceLog();
debounceLog();
// log tada only 1 time
```

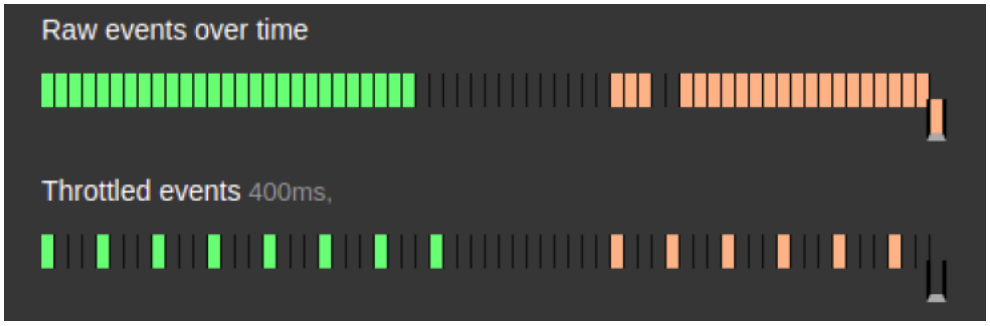
4. lodash.debounce

```
# npm
npm i lodash.debounce

# or yarn
yarn add lodash.debounce
```

```
import debounce from 'lodash.debounce';

function log() {
  console.log('tada');
}

log();
log();
log();
log();
log();
// log tada 5 times

const debounceLog = debounce(log, 500);
debounceLog();
debounceLog();
debounceLog();
debounceLog();
debounceLog();
// only log tada ONCE
```

> lodash.debounce: https://lodash.com/docs/4.17.15#debounce

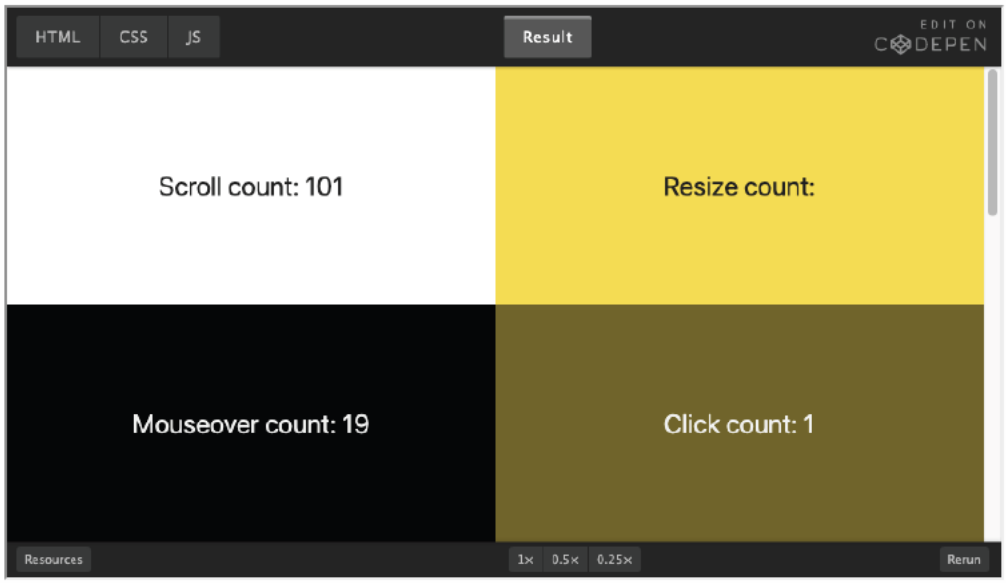
▼ **150. Throttle Technique**



> Src: https://blog.knoldus.com/how-to-optimize-web-app-with-debounce-and-throttle/

1. Throttle demo

- Usually throttle is applied to scroll to prevent calling function too many times.



> Src: https://webdesign.tutsplus.com/tutorials/javascript-debounce-and-throttle--cms-36783

2. How it works

- throttle = trigger max 1 call every period of time

```
function throttle(callback, wait) {}

function log() {}
const throttleLog = throttle(log, 500);
throttleLog();
throttleLog();
throttleLog();
```

3. Simple Implementation

```
function throttle(callback, wait) {
  let isThrottling = false;

  return function () {
    if (isThrottling) return; // 2, 3, 4 come here

  isThrottling = true; // 1 come here
  setTimeout(() => {
    callback();

    isThrottling = false;
  }, wait);
  };
}

const throttledLog = throttle(log, 500);
// 0 -> 500
throttledLog(); // set timeout 500 --> call --> turn of flag throttle
throttledLog(); // check throttling --> ignore
throttledLog(); // check throttling --> ignore
throttledLog(); // check throttling --> ignore

// 600 --> 1100
setTimeout(throttledLog, 600); // start throttling
setTimeout(throttledLog, 600);
setTimeout(throttledLog, 600);
setTimeout(throttledLog, 600);
setTimeout(throttledLog, 700);
setTimeout(throttledLog, 700);
```

5. debounce vs throttle

	debounce	throttle
concept	call after a period of time receving no trigger	trigger max 1 call every period of time
reduce number of calls	YES	YES
common case	search when stop typing	do something on scroll

▼ **151. setTimeout with "this"**

1. Issues

- Can not bind this to setTimeout.
- `this` in callback function of setTimeout default is a global object (window)

```
const student = {
  name: 'Easy Frontend',

  sayHello() {
    console.log('NAME', this.name);
    console.log('THIS', this);
  },
}

student.sayHello(); // 'Easy Frontend'

setTimeout(student.sayHello);
// this is window, not student

setTimeout.call(student, student.sayHello);
// Uncaught TypeError: Illegal invocation
```

2. Solution: use a wrapper function

```
const student = {
  name: 'Easy Frontend',

  sayHello() {
    console.log('NAME', this.name);
    console.log('THIS', this);
  },
}

setTimeout(() => {
  student.sayHello(); // this is student --> 'Easy Frontend'
});
```

3. Solution: use bind()

```
const student = {
  name: 'Easy Frontend',

  sayHello() {
    console.log('NAME', this.name);
    console.log('THIS', this);
  },
}
```

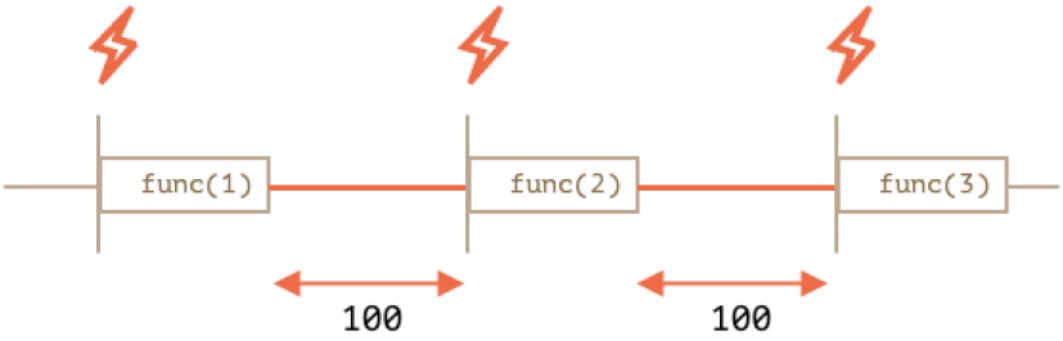
```
setTimeout(student.sayHello.bind(student)); // 'Easy Frontend'
```

Some examples

```
const student = {
  name: 'Easy Frontend',

  normal() {
    console.log('NAME', this.name);
    console.log('THIS', this);
  },

  arrow: () => {
    console.log('NAME', this.name);
    console.log('THIS', this);
  },

  timeoutNormal() {
    console.log('OUTER THIS', this);

    setTimeout(function() {
      console.log('NAME', this.name);
      console.log('THIS', this);
    })
  },

  timeoutArrow() {
    console.log('OUTER THIS', this);

    setTimeout(() => {
      console.log('NAME', this.name);
      console.log('THIS', this);
  })
  }
}
```

> Reference: https://developer.mozilla.org/en-US/docs/Web/API/setTimeout#the_this_problem

▼ 152. setInterval(callback, timeout)



> Reference: https://javascript.info/settimeout-setinterval

1. setInterval Overview

- **What**: exec callback every period of time until it's cleared by clearInterval

- **When**: do something repeatly
- **Where**: working on both browser + nodejs

```
const intervalId = setInterval(callback, delay);
// callback is the function will be executed once the timer is expired
// delay is the time in milliseconds. 1 second = 1000 milliseconds

// intervalId is a positive integer, used to identify an interval
// and clear it by using clearInterval(intervalId)
```

2. clearInterval

```
const intervalId = setInterval(() => {
  console.log('Hey, are you there?');
}, 2000);

clearInterval(intervalId);
```

💡 **NOTE**: setInterval() and setTimeout() share the same pool of IDs.

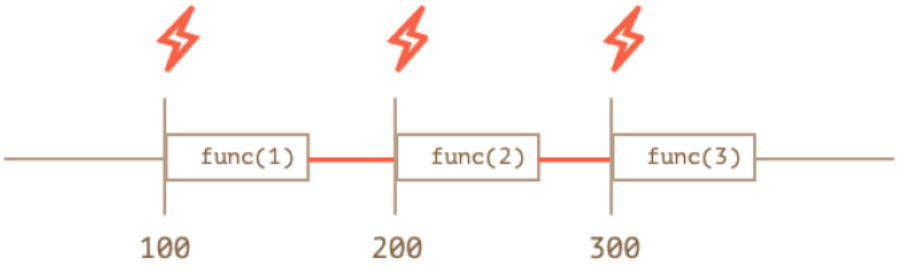
3. Example - Countdown

```
function countdown(seconds) {
  let currentSecond = seconds;

  const intervalId = setInterval(() => {
    console.log(currentSecond);

    if (currentSecond <= 0) {
      clearInterval(intervalId);
    }

    currentSecond -= 1;
  }, 1000);
}

countdown(10);
```

4. setTimeout and setInterval

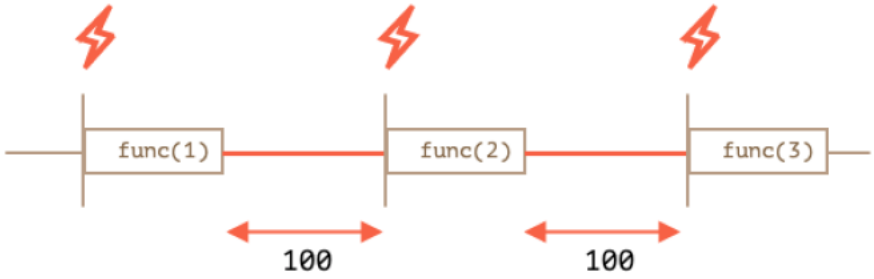
```
let i = 1;
setInterval(function() {
  func(i++);
}, 100);
```

For `setInterval` the internal scheduler will run `func(i++)` every 100ms:



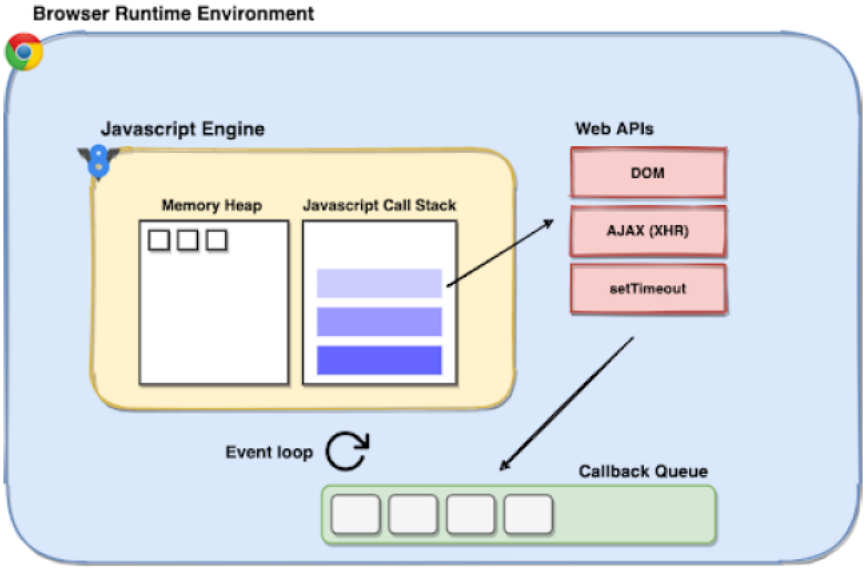

```
setTimeout(function run() {
func(i++);
  setTimeout(run, 100);
}, 100);
```

And here is the picture for the nested `setTimeout`:



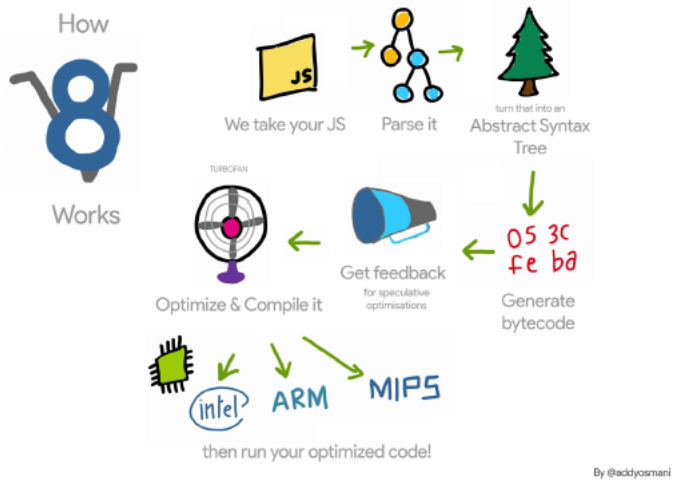
> Src: https://javascript.info/settimeout-setinterval

▼ 153. Event L0oop - Browser



> Src: https://scoutapm.com/blog/async-javascript

1. Javascript Engine



> Src: https://dzone.com/articles/v8-javascript-engine-the-non-stop-improvement

#	Name	Desc
1	Javascript Engine	computer program that exec js code
2	Ecmascript Engine	computer program that exec code that implements ECMAScript
3	Compiler	AoT (Ahead of Time), compile all -> exec
4	Interpreter	line by line, read and exec
5	Just-in-time (JIT) compiler	use the best of Compiler + Interpreter (modern browsers)

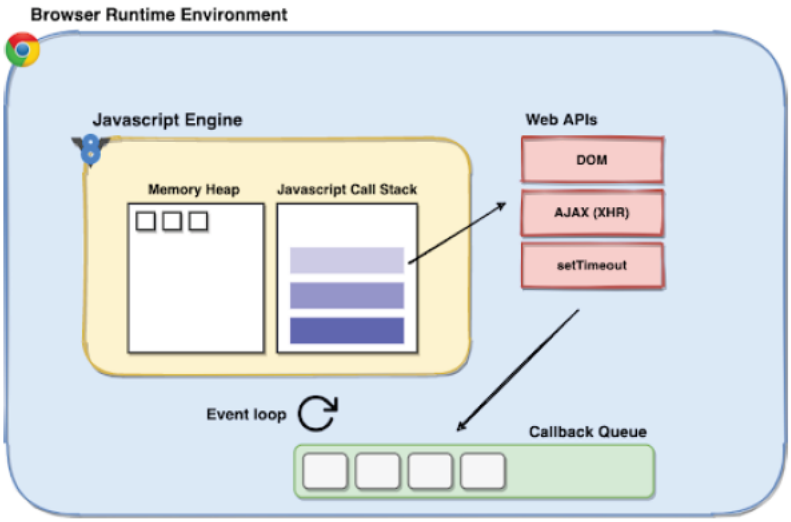
> Src: https://hacks.mozilla.org/2017/02/a-crash-course-in-just-in-time-jit-compilers/

ECMAScript Engines

#	Environment	Engine
1	Chrome	V8 - Google open source
2	NodeJS	V8
3	Microsoft Edge	V8 (v79)
4	Firefox	Spider Monkey
5	Safari	JavascriptCore

> Src: https://en.wikipedia.org/wiki/List_of_ECMAScript_engines

2. Event Loop Components

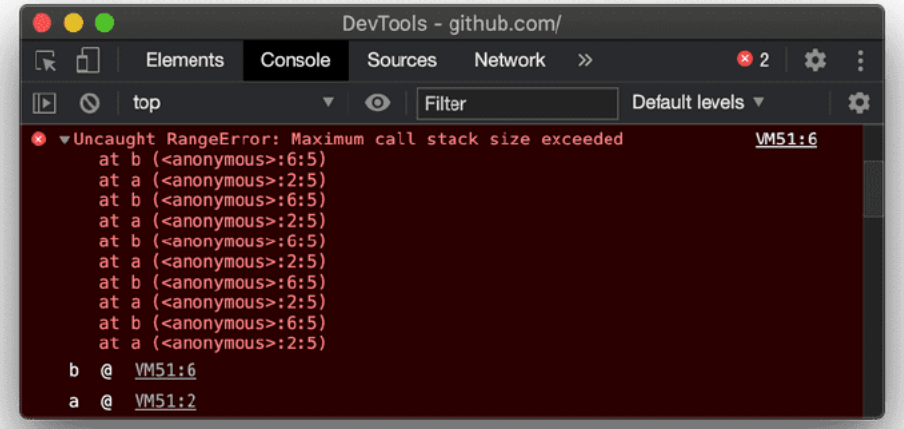


> Src: https://scoutapm.com/blog/async-javascript

#	Name	Desc
1	Heap	where to store objects and functions read more
2	Call Stack	keep track of the functions that a script calls
3	Web API	APIs of web browsers to help you make AJAX request, DOM manipulation, do things concurrently, ...
4	Callback Queue	run code after the execution of the Web API call has finished
5	Event Loop	run-to-complete, add call from Queue when the call stack is empty.

> Reference: https://felixgerschau.com/javascript-memory-management/

💡 Maximum call stack size exceeded - 10k -> 50k



> Src: https://felixgerschau.com/javascript-event-loop-call-stack/

Callback Queue vs Promise Queue (Macrotask vs Microtask)

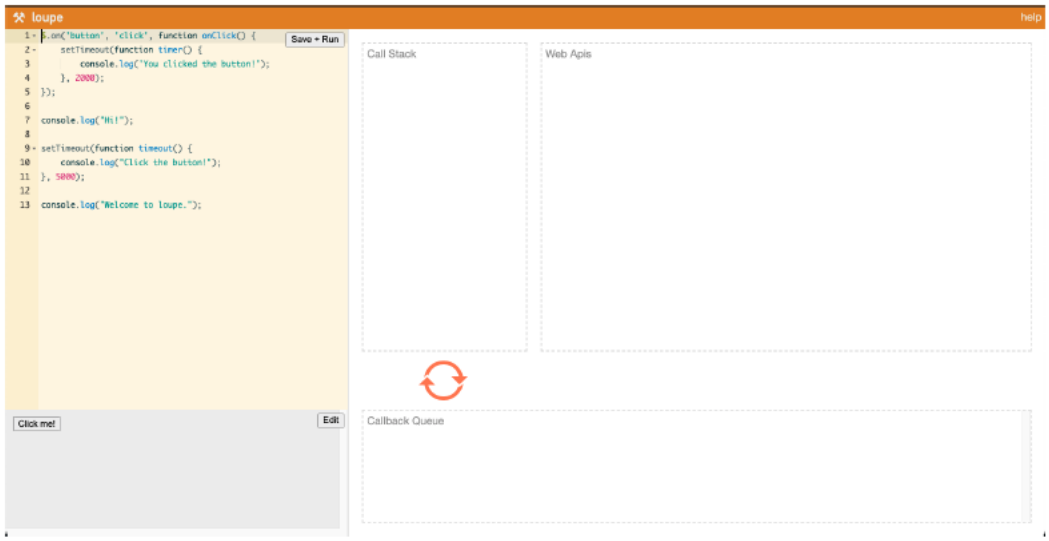
- Promise Queue has higher priority than callback queue
- Or we can say: Microtask has higher priority than macrotask

```
console.log('a');

setTimeout(() => console.log('b'), 0);

new Promise((resolve, reject) => {
  resolve();
})
  .then(() => {
  console.log('c');
});
```

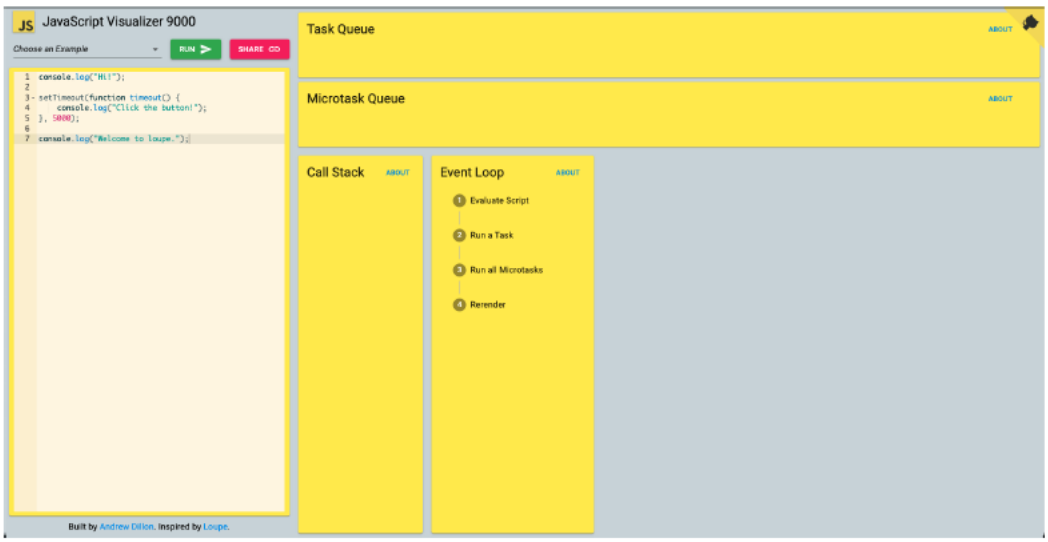
```
console.log('d');
// a -> d -> c -> b
```

3. Event Loop Visualizer

http://latentflip.com/loupe

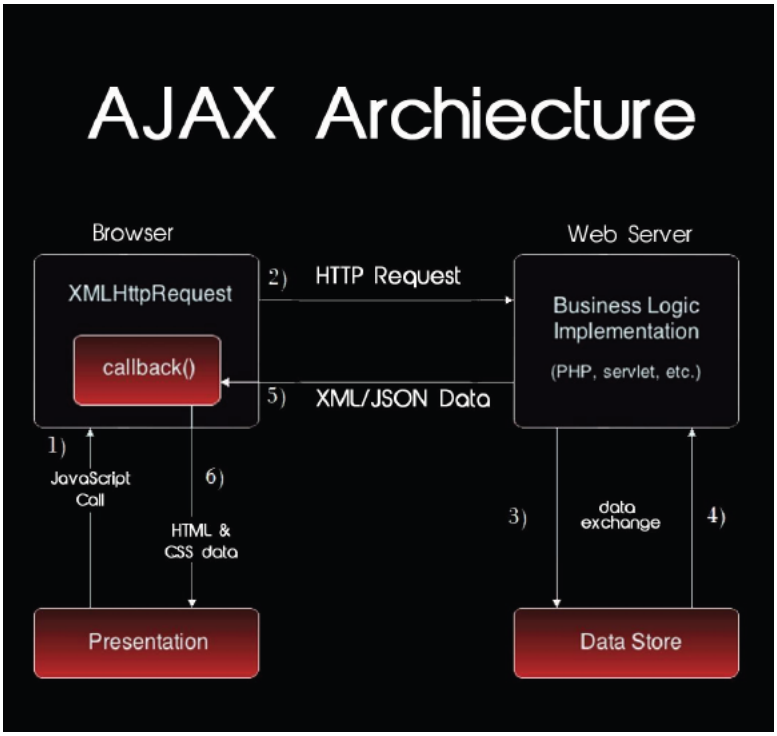


https://www.jsv9000.app/



▼ **155. Ajax - Asynchronous JavaScript and XML**

1. Ajax Overview



> Src: https://in.pinterest.com/pin/849913760926810878/

Ajax is made up of the following technologies:

- XHTML and CSS for presenting information
- Document Object Model (DOM) for dynamically interacting with and displaying the presented information
- XMLHttpRequest object to manipulate data asynchronously with the web server
- XML, HTML, and XSLT for data interchange and manipulation
- JavaScript for binding data requests and information display

2. HTTP APIs

An HTTP API is an API that uses Hypertext Transfer Protocol as the communication protocol between the two systems.

#	Name	Desc
1	Most Popular REST API	Representational State Transfer and is an architectural pattern for creating web services
2	GraphQL API	GraphQL API allows requests to call for the exact amount of data and type it needs

> Src: https://www.educative.io/blog/what-are-rest-apis

3. REST API

Not all HTTP APIs are REST APIs. They need to meet the following requirements:

- client-server
- stateless server

- cachable
- **uniform interface**
- layered system

> Src: https://www.educative.io/blog/what-are-rest-apis

Method	Resource	Has payload	Desc
GET	/students	NO	Get all students
GET	/students/:studentId	NO	Get one student by id
POST	/students	YES	Add new student
PUT	/students/:studentId	YES	Full update of a student
PATCH	/students/:studentId	YES	Paritial update of a student
DELETE	/students/:studentId	NO	Remove a student by id

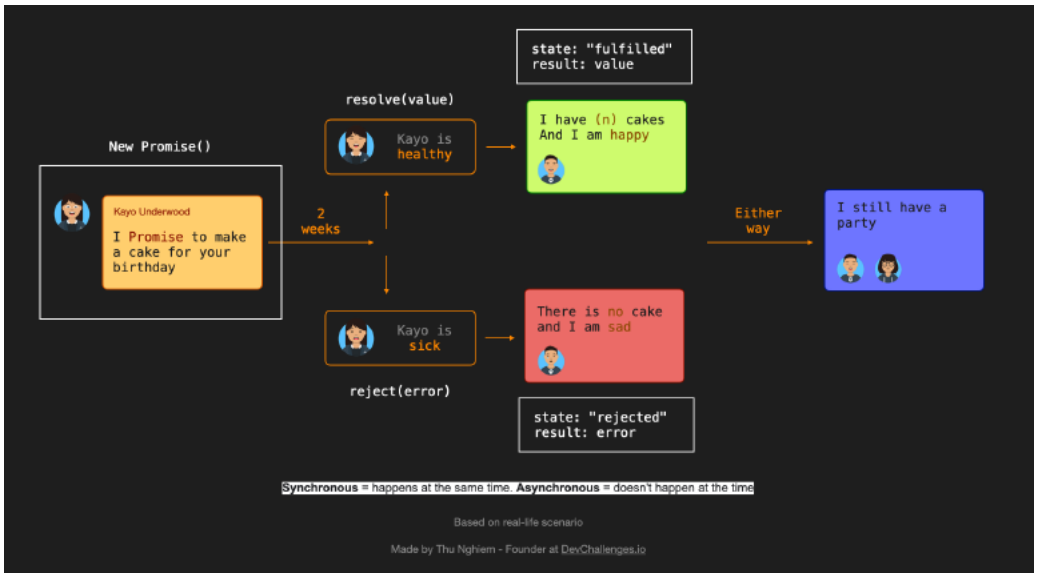
> Src: https://restfulapi.net/resource-naming/

4. XMLHttpRequest

```
const url = 'http://js-post-api.herokuapp.com/api/students?_page=1'
const xhr = new XMLHttpRequest();

xhr.open('GET', url);
xhr.responseType = 'json';

xhr.addEventListener('load',function() {
  console.log(xhr.response);
});

xhr.addEventListener('error', function() {
  console.log('Something wrong :P')
});

xhr.send();
```

> Src: https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest

▼ 156. Promise



> Src: https://www.freecodecamp.org/news/learn-promise-async-await-in-20-minutes/

1. Callback hell

```
// Callback Hell


a(function (resultsFromA) {
    b(resultsFromA, function (resultsFromB) {
        c(resultsFromB, function (resultsFromC) {
            d(resultsFromC, function (resultsFromD) {
                e(resultsFromD, function (resultsFromE) {
                    f(resultsFromE, function (resultsFromF) {
                        console.log(resultsFromF);
                    })
                })
            })
        })
    })
});

```

> Src: https://medium.com/@jaybhoyar1997/avoiding-callback-hell-in-node-js-7c1c16ebd4d3

2. Promise Overview

Promise states

	Desc
pending	initial state
fulfilled	operation completed successfully
rejected	opeation failed

> Src: https://scoutapm.com/blog/async-javascript

Create a new promise

```
> const promise = new Promise();
  console.log(promise);
▶ Uncaught TypeError: Promise resolver undefined is not a function
      at new Promise (<anonymous>)
      at <anonymous>:1:17
```

```
const promise = new Promise((resolve, reject) => {});
console.log(promise);
```

```
> const promise = new Promise((resolve, reject) => {});
  console.log(promise);
▼ Promise {<pending>}                                    react_devtools_backend.js:4049
  ▶ [[Prototype]]: Promise
    [[PromiseState]]: "pending"
    [[PromiseResult]]: undefined
```

Retrieve promise value

```
const promise = new Promise((resolve, reject) => {
  setTimeout(() => resolve(1), 3000);
});

promise
  .then(result => console.log(result))
  .catch(error => console.log(error));
```

```
const promise = new Promise((resolve, reject) => {
  reject(new Error('oops, something wrong!!!'))
});

promise
  .then(result => console.log(result))
  .catch(error => console.log(error));
```

💡 **GOLDEN RULE**: Always handle error when using promise.

```
> const promise = new Promise((resolve, reject) => {
    reject(new Error('oops, something wrong!!!'))
  });

  promise.then(result => console.log(result))
<· Promise {<rejected>: Error: oops, something wrong!!!
      at <anonymous>:2:10
      at new Promise (<anonymous>)
      at <an…}
⊗ ▸Uncaught (in promise) Error: oops, something wrong!!!
      at <anonymous>:2:10
      at new Promise (<anonymous>)
      at <anonymous>:1:17
```

Otherwise, you will get uncaught (in promise) error as captured above.

3. Promise Methods

Static methods

#	Name	Desc
1	Promise.resolve(value)	returns a promise, `value` can be a promise or not
2	Promise.reject()	return a promise with rejected reason
3	Promise.all()	Wait for all promises to be resolved, or for any to be rejected.
4	Promise.allSettled()	Wait until **all promises have settled** (each may resolve or reject).
5	Promise.any()	Wait until any of the promises is **fulfilled**.
6	Promise.race()	Wait until any of the promises is **fulfilled or rejected**.

```
const b = new Promise((resolve) => {
  resolve('BBB');
})

const promiseA = Promise.resolve('AAA');
const promiseB = Promise.resolve(b);

Promise.all([promiseA, promiseB])
  .then(([resultA, resultB]) => {
    console.log(resultA, resultB)
  })
  .catch(error => console.log(error))
```

Instance methods

#	Name	Desc
1	then	return a new promise, happens when promise is fulfilled
2	catch	catch error if promise is rejected
3	finally	go here no matter the promise is fulfilled or rejected

```
Promise.resolve('tada')
  .then(message => console.log(message))
  .catch(error => console.log(error))
  .finally(() => {
    // usually we hide loading here
})
```

4. Promise Chaning

```
Promise.resolve(5)
  .then(n => n * 2)
  .then(n => Promise.resolve(n * 2))
  .then(n => {
    const finalNumber = n + 10;
    console.log(finalNumber);

    return finalNumber;
  })
  .catch(error => console.log(error));
```

```
fetch('https://js-post-api.herokuapp.com/api/students?_page=1')
  .then(response => response.json())
  .then(data => console.log(data))
  .catch(error => console.log(error));
```

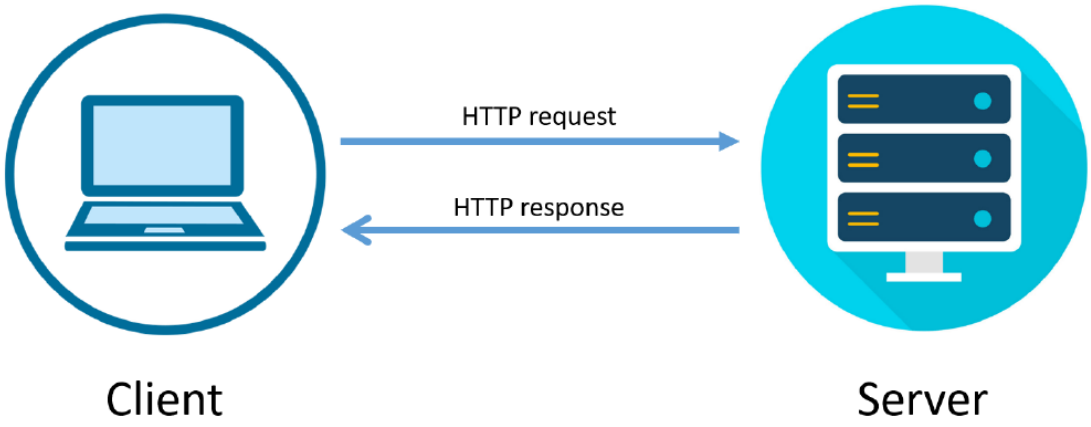
5. Fake API

Sometimes, the API from Backend is not ready, you need to fake it your self first to not block your works.

```
// studentApi.js
const studentApi = {
  getAll() {
    return new Promise((resolve) => {
      setTimeout(() => {
        resolve({
          data: [
            { id: 1, name: 'Alice' },
            { id: 2, name: 'Bob' },
          ],
          pagination: {
            _total: 2,
            _page: 1,
            _limit: 10,
          }
        })
      }, 1000);
    })
  }
}
```

```
// app.js
studentApi.getAll()
  .then(response => console.log(response))
  .catch(error => console.log(error));
```

▼ 157. fetch()



> Src: https://bytesofgigabytes.com/networking/how-http-request-and-response-works/

1. fetch Overview

- The Fetch API provides an interface for fetching resources.
- Web APIs -> global method **fetch()**.

fetch

- it starts the process of fetching a resource from the network
- returns a promise which fulfilled once the response is available
- it only rejects when a network error is encountered
- it doesn't reject on HTTP errors (4xx, 5xx)
- you can check **Response.ok** to handle error properly

```
const url = 'https://js-post-api.herokuapp.com/api/students?_page=1';
const init = {
  method: 'POST', // GET, PUT, PATCH. DELETE
  headers: {
    'Content-Type': 'application/json',
    Authorization: 'Bearer YOUR_TOKEN_HERE'
  },
  body: JSON.stringify({ name: 'Easy Frontend' }),
}

const promise = fetch(url, init);
```

Init object

#	Name	Desc
1	method	GET, POST, PUT, PATCH, DELETE, ...
2	headers	'Content-Type', Authorization, x-*
3	body	URLSearchParams, FormData, Blob, ...
4	signal	an AbortSignal used to abort if desired read more

> Signal reference: https://developer.mozilla.org/en-US/docs/Web/API/AbortSignal

Common Content-Type

Content-Type	Desc
application/json	JSON data (most of the time)
multipart/form-data	FormData (upload file: images, pdf, ...)
application/x-www-form-urlencoded	key-value on URL

> Src: https://developer.mozilla.org/en-US/docs/Web/API/fetch#parameters

```
fetch('https://js-post-api.herokuapp.com/api/students?_page=1')
  .then(response => response.json())
  .then(data => console.log(data));
```

2. fetch with headers / body

- GET, DELETE: not use body
- POST/PUT/PATCH: send with body

```
// Add new student
fetch('https://js-post-api.herokuapp.com/api/students', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
  },
  body: JSON.stringify({
    name: 'Easy Frontend',
    age: 18,
    mark: 9,
    gender: 'male',
  })
}).then(...).catch(...)
```

```
// Update partial info of student
fetch('https://js-post-api.herokuapp.com/api/students/e0df2354-1014-4f40-97c4-76e1dac88ab5', {
  method: 'PATCH',
  headers: {
    'Content-Type': 'application/json',
  },
  body: JSON.stringify({
    name: 'Easy',
    age: 20,
  })
}).then(...).catch(...)
```

```
// Delete a student
fetch('https://js-post-api.herokuapp.com/api/students/e0df2354-1014-4f40-97c4-76e1dac88ab5', {
  method: 'DELETE',
  headers: {
    'Content-Type': 'application/json',
  },
}).then(...).catch(...)
```

> Reference: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers

3. Handle Errors

```
fetch('https://js-post-api.herokuapp.com/api/invalid-endpoint', {
  method: 'GET',
  headers: {
    'Content-Type': 'application/json',
    },
  })
    .then(response => {
      if (response.ok) return response.json();

      // TODO: How you handle errors here? --> It depends on what your API returns
      // Solution 1: return Promise.reject(new Error('Something wrong!'));
      // Solution 2: throw new Error(response.statusText);
      return response.json().then(data => {
        throw new Error(data?.message || 'Something went wrong!');
      })
    })
    .catch(error => {
      console.log(error);
      // Toast message
      // Send report to log server (Sentry)
  })
```

Response status codes

Status Code	Name
1xx	Informational responses
2xx	Successful responses
3xx	Redirects
4xx	Client errors
5xx	Server errors

Common status code

Status code	Name
200	OK
201	Created
301	Moved Permanently
400	Bad Request
401	Unauthorized
403	Forbidden
404	Not Found
429	Too Many Requests
500	Internal Server Error
502	Bad Gateway

> Reference: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

💡 What to do when you get an error from API
- Check URL
- Check payload
- Check headers
⇒ Then come to ask BE with the above information

▼ 159. CSP and CORS

1. CSP - Content Security Policy

> Src: https://www.rahulpnath.com/blog/http-content-security-policy-csp/

```
// try to run this line of code on MDN and see
fetch('http://js-post-api.herokuapp.com/api/students?_page=1')
```

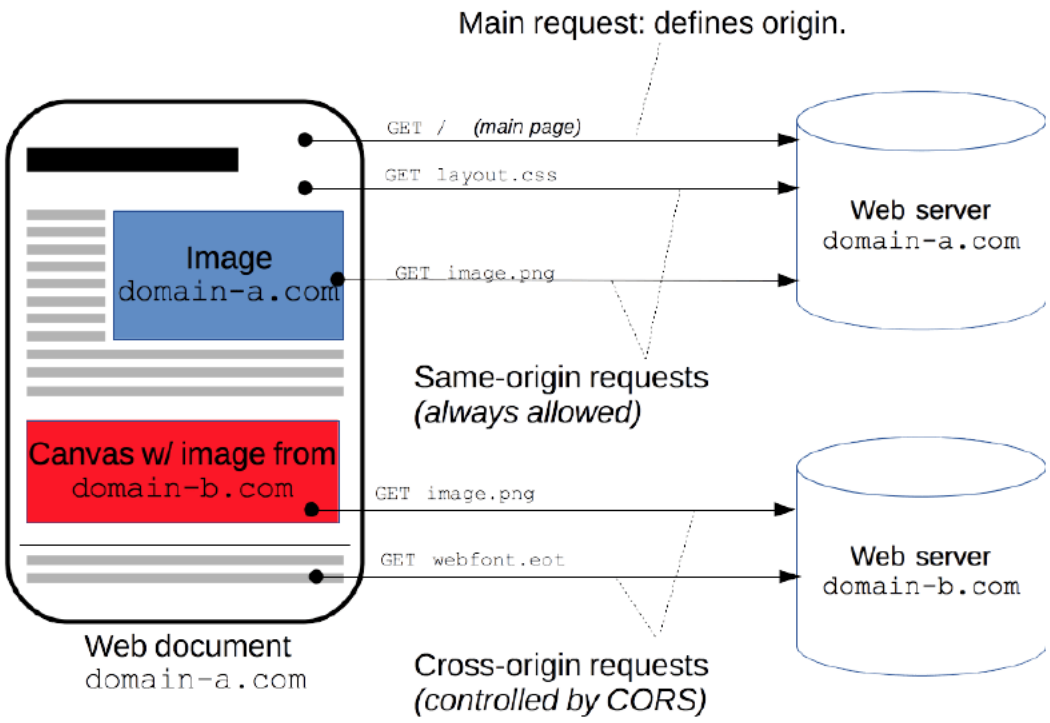
```
// error: Refused to connect to 'http://js-postapi.
herokuapp.com/api/students?_page=1' because it violates the document's
Content Security Policy
```

> Src: https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP

2. CORS - Cross - Origin Resoure Sharing

```
Access to fetch at 'https://joke-api-strict-cors.appspot.com/r localhost/:1
andom joke' from origin 'http://localhost:3000' has been blocked by CORS
policy: No 'Access-Control-Allow-Origin' header is present on the requested
resource. If an opaque response serves your needs, set the request's mode
to 'no-cors' to fetch the resource with CORS disabled.
```

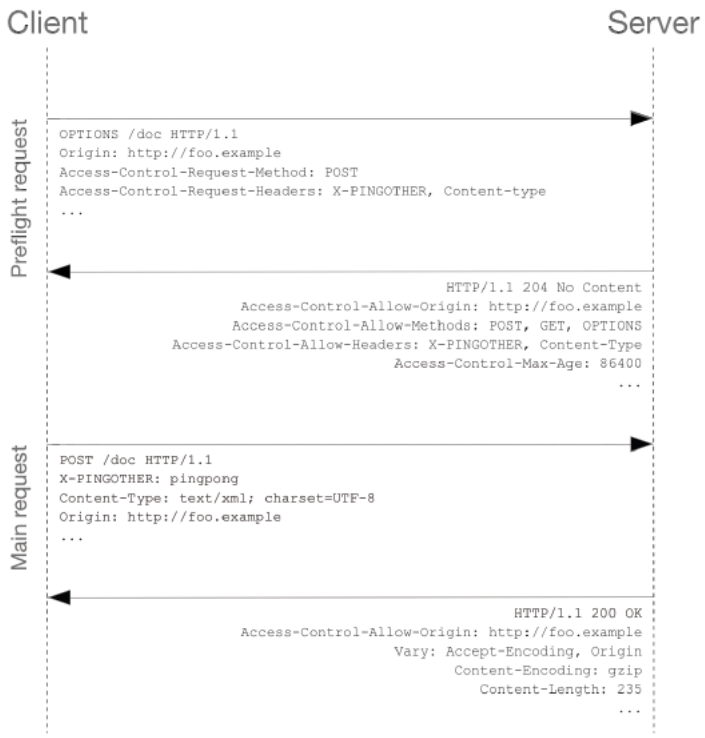
> Src: https://medium.com/@dtkatz/3-ways-to-fix-the-cors-error-and-how-access-control-allow-origin-works-d97d55946d9



> Reference: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS

```
// access this from google.com console
fetch('https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS/bcd.json')

/*
Access to fetch at 'https://developer.mozilla.org/en-
US/docs/Web/HTTP/CORS/bcd.json' from origin 'https://www.google.com' has
been blocked by CORS policy: No 'Access-Control-Allow-Origin' header is
present on the requested resource. If an opaque response serves your
needs, set the request's mode to 'no-cors' to fetch the resource with CORS
```

```
disabled.
*/
```



▼ 160. Async Function

1. Async Funciton Syntax

```
// Async function declaration
async function func1() {}

// Async function expression
const func2 = async function () {};

// Async arrow function
const func3 = async () => {};

// Async method definition in an object literal
const obj = { async say() {} };
```

await should be used in **async function**

```
// work
async function fetchData() {
  const data = await studentApi.getAll();
  console.log(data);
}
// error
function fetchData() {
  const data = await studentApi.getAll();
  console.log(data);
}
```

async function **always** return Promise

```
async function getNumber() {
  return 10;
}

getNumber(); // it returns a promise 😉
```

```
async function getNumber() {
  return Promise.resolve(10);
}
getNumber(); // it returns a promise
```

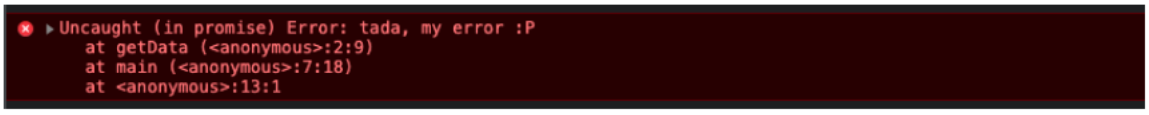
2. Handle Errors

💡 **IMPORTANT**: always handle errors when using Promise / calling APIs

```
async function getAllStudent() {
  try {
    const url = 'http://js-post-api.herokuapp.com/api/students?_page=1';
    const response = await fetch(url);
    const data = await response.json();

    console.log(data);
  } catch (error) {
    console.log(error);
  }
}
```

DON'T FORGET await keyword 🔥

```
async function getData() {
  // return Promise.reject(new Error('tada, my error :P'))
  throw new Error('tada, my error :P')
}
async function main() {
  try {
    const data = getData();
  } catch (error) {
    console.log(error);
  }
}

main();
```



3. Parallel and Sequence APIs

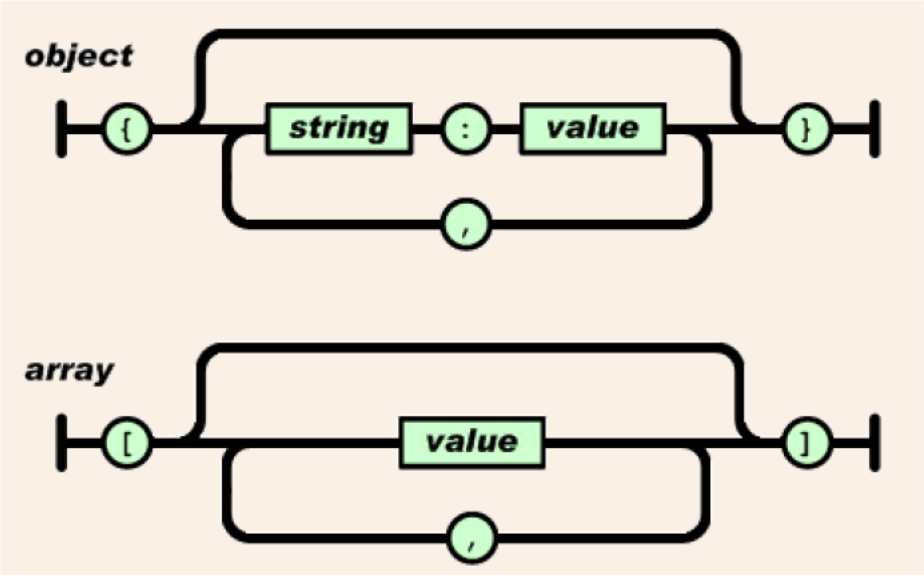
- Parallel: use Promise.all()
- Sequence APIs: use async/await

```
const promiseList = [
  getAllStudent,
  getAllCategories,
  getAllCities,
];

// Trigger API calls at the same time
```

```
Promise.all(promiseList)
  .then(([studentList, categoryList, cityList]) => {
    console.log(studentList, categoryList, cityList);
  })
  .catch(error => console.log(error));
```

> Src: https://exploringjs.com/impatient-js/ch_async-functions.html

▼ **162. JSON - JavaScript Object Notation**



> Src: https://twiki.org/cgi-bin/view/Blog/BlogEntry201503x1

1. What is JSON

- JSON is standard built-in objects
- JSON is a global object
- JSON is a syntax for serializing:
  - objects
  - arrays
  - numbers
  - strings
  - booleans
  - null

2. JSON Methods

#	Name	Desc
1	JSON.parse(text)	JSON -> Javascript value
2	JSON.stringify(value)	Javascript value -> JSON string

3. JSON.stringify()

```
const student = { id: 1, name: 'Easy Frontend', age: undefined }
JSON.stringify(student);
// '{"id": 1, "name": "Easy Frontend" }'

const list = [null, undefined, NaN];
JSON.stringify(list);
// [null, null, null]
```

> Src: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON/stringify

4. JSON.parse()

- Property name MUST BE double quoted
- For numbers, leading zeros are prohibited
- NaN and Infinity are unsupported
- Not allow trailing commas

```
JSON.parse('123'); // 123
JSON.parse('true'); // true
JSON.parse('null'); // null
JSON.parse('{}'); // {}
JSON.parse('[]'); // []

JSON.parse('0123'); // Syntax Error
JSON.parse('NaN'); // Syntax Error
JSON.parse('Infinity'); // Syntax Error

// property name must use double quotes
JSON.parse("{ 'id': 123 }"); // Syntax Error
JSON.parse('{ "id": 123 }'); // { id: 123 } it works

// Not allow trailing commas
JSON.parse('[1, 2, 3, 4, ]');
JSON.parse('{"foo" : 1, }');
```

> Src: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON/parse#json.parse_does_not_allow_tra

▼ SECTION 17: API AND POSTMAN

▼ **165. API Documentation**

1. Roles

Role	Responsible
FE	Integrate API according to document
BE	Create API + provide API document

2. Environments

#	Name	Desc
1	Local / Dev	for local development
2	Staging / QA	where QA can jump in to test
3	Pre-produciton (less common)	similar to prod, a step to double check the configs
4	Production / User	LIVE - user is using this version

3. API Documentation

#	Name	Desc
1	Chat / Document	chat via slack, send a word document, ...
2	Postman Collection	shared a postman collection to import
3	Swagger	easy swagger
4	Public API document	Stripe API

> Reference: https://app.swaggerhub.com/apis-docs/paulnguyen.mn/easy-javascript/1.0.0

> Reference: https://stripe.com/docs/api/payment_methods/update

▼ 166. How to use APIs

> Reference JSON SERVER: https://github.com/typicode/json-server#plural-routes

1. Student APIs

Method	Resource	Has payload	Desc
GET	/students	NO	Get all students
GET	/students/:studentId	NO	Get one student by id
POST	/students	YES	Add new student
PUT	/students/:studentId	YES	Full update of a student
PATCH	/students/:studentId	YES	Paritial update of a student
DELETE	/students/:studentId	NO	Remove a student by id

2. Post APIs

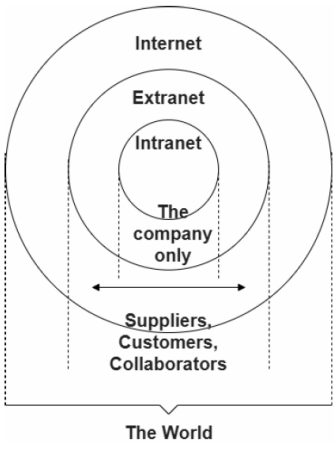
Method	Resource	Has payload	Desc
GET	/posts	NO	Get all posts
GET	/posts/:postId	NO	Get one post by id
POST	/posts	YES	Add new post
PUT	/posts/:postId	YES	Full update of a post
PATCH	/posts/:postId	YES	Paritial update of a post
DELETE	/posts/:postId	NO	Remove a post by id

▼ SECTION 18: BROWER AND WEB APIS

▼ **170. Browser Overview**

> Src: https://www.gizmoarc.com/most-popular-internet-browsers-from-2003-to-2020/

1. What is Browser?



- Internet: large network of computers which communicate all together.
- Browser: a program that retrieves and displays pages from the Web, and lets users access further pages through hyperlinks.
- WWW/W3/Web: an interconnected system of public webpages accessible through the Internet. Includes: HTTP + URI + HTML.
- List of browsers

2. Common Browsers

> Src: https://gs.statcounter.com/browser-market-share

3. Cross Browsers Testing

- CrossBrowserTesting
- BrowserStack

> Reference: https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Cross_browser_testing/Introduction

Vendor prefix

#	Name	Browser
1	-webkit-	Chrome, Safari, newer versions of Opera, all iOS browsers, ...
2	-moz-	Firefox
3	-o-	pre-webkit versions of Opera
4	-ms-	Internet Explorer and Microsoft Edge

> Src: https://developer.mozilla.org/en-US/docs/Glossary/Vendor_Prefix

4. Global Object: window

- A global variable, window, representing the window in which the script is running, is exposed to JavaScript code.
- Each tab has its own window object.

> Src: https://developer.mozilla.org/en-US/docs/Web/API/Window

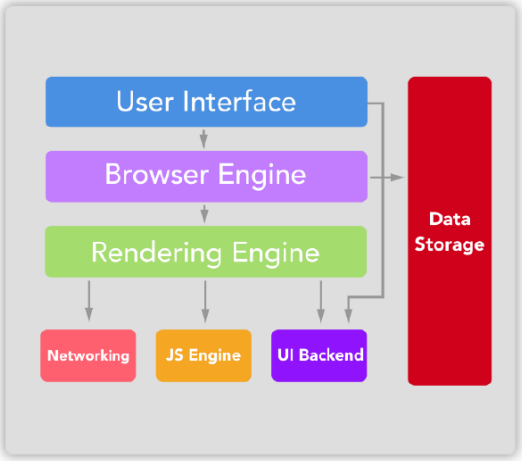
▼ 171. Browser Components

1. Browser Engines

Engine	Status[a]	Steward	License	Embedded in
WebKit	Active	Apple	GNU LGPL, BSD-style	Safari browser, plus all browsers for iOS[4]
Blink	Active	Google	GNU LGPL, BSD-style	Google Chrome and all other Chromium-based browsers, notably Microsoft Edge and Opera
Gecko	Active	Mozilla	Mozilla Public	Firefox browser and Thunderbird email client
Goanna	Active	M. C. Straver[5]	Mozilla Public	Pale Moon and Basilisk browsers, K-Meleon browser beginning with version 76.2G
Flow	Active	Ekioh[6]	Proprietary	Flow browser[7]
Trident[b]	Maintained	Microsoft	Proprietary	Internet Explorer browser
EdgeHTML	Maintained	Microsoft	Proprietary	UWP apps; formerly in the Edge browser[9]
KHTML	Maintained	KDE	GNU LGPL	Konqueror browser
Servo	Maintained	Linux Foundation	Mozilla Public	experimental browsers[10][11]
NetSurf[c]	Maintained	hobbyists[13]	GNU GPLv2	NetSurf browser[14]
Presto	Discontinued	Opera	Proprietary	formerly in the Opera browser

> Src: https://en.wikipedia.org/wiki/Comparison_of_browser_engines

2. Browser Components

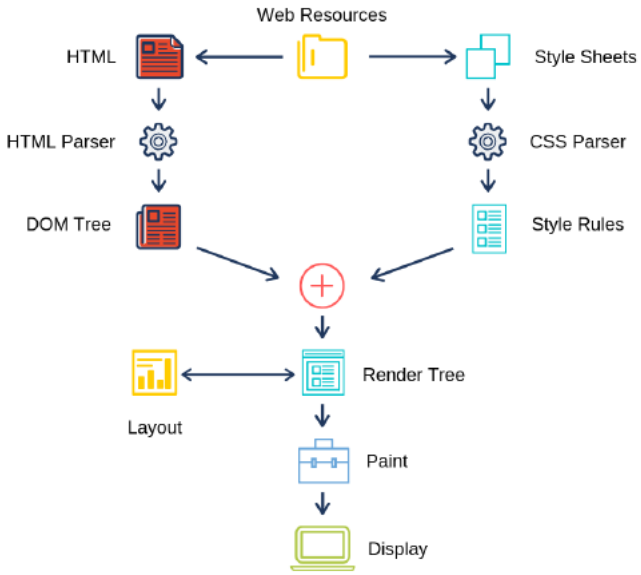


> Src: https://codeburst.io/how-browsers-work-6350a4234634

#	Name	Desc
1	User Interface	Address Bar, Home, Back buttons, ...
2	Browser Engine	Receive action from UI, then works with Storage + Rendering Engine
3	Rendering Engine	Display requested content (HTML/CSS), or more (pdf, xml) with plugins
4	Networking Component	Handle network tasks (HTTP, WebSocket, WebRTC)
5	Javascript Engine	Javascript Engine (V8, SpiderMonkey, ...)
6	UI Backend	Used for drawing everything (UI + in the rendering engine), use OS interface (checkbox, dropdown, ...)
7	Data Storage	Used for persisting data locally (http cookies, browser caching, web storage: local storage, session storage and IndexedDB)

3. Rendering Engine

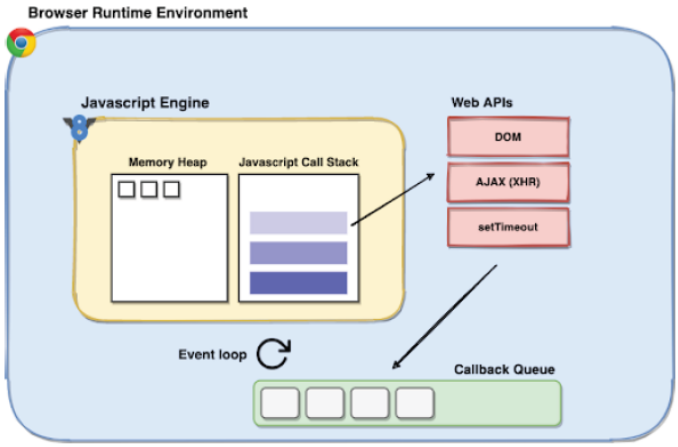
Read more about: Repaint and Reflow



> Src: https://www.bitcoininsider.org/article/66724/how-do-web-browsers-work

🌐 **REFERENCES**
https://codeburst.io/how-browsers-work-6350a4234634
https://www.browserstack.com/guide/browser-rendering-engine
https://www.html5rocks.com/en/tutorials/internals/howbrowserswork/
https://blog.sessionstack.com/how-javascript-works-the-rendering-engine-and-tips-to-optimize-its-performance-7b95553baeda

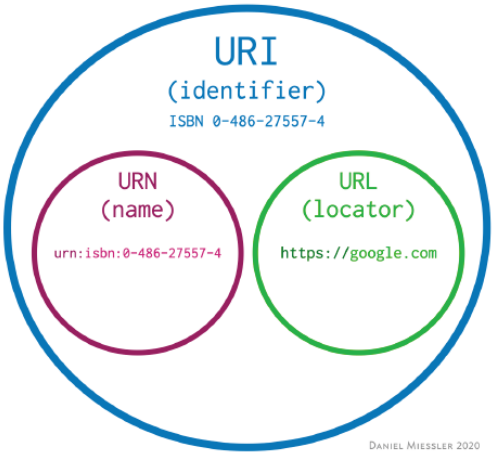
▼ 172. Web APIs



> Src: https://scoutapm.com/blog/async-javascript

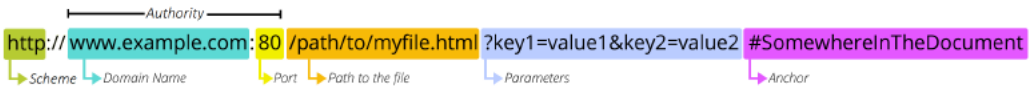
#	Name	Desc
1	Canvas API	game, animation, ... read more
2	Clipboard API	copy / paste read more
3	Console API	read more
4	DOM API	section 19 read more
5	Fetch API	section 16 read more
6	Geolocation API	get current location read more
7	HTML Drag and Drop API	read more
8	History API	navigate to URL programmatically read more
9	Server Sent Events	one way communication SERVER -> CLIENT read more
10	The WebSocket API	two ways communication SERVER <-> CLIENT read more
11	Service Workers API	offline, caching, web notifications, ... read more
12	Storage	local storage, session storage, cookies, ... read more
13	URL API	read more
14	XMLHttpRequest	section 16 read more

▼ **173. URL**



> Src: https://danielmiessler.com/study/difference-between-uri-url/

1. What is URL?



> Src: https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_is_a_URL

Create a new URL object

```
const url = new URL('https://ezfrontend.com/blog?page=1&limit=10#title');
console.log(url);
```

2. URLSearchParams

```
const params = new URLSearchParams('?_page=1&_limit=10');
// leading question mark will be ignored
```

#	Name	Desc
1	get(key)	get value of a **key**, return **null** if not found
2	set(key, value)	add new/override value of a key
3	delete(key)	remove a key
4	append(key, value)	add new key/value (even it's existed already)
4	has(key)	check if key is available, return true/false
5	toString()	convert to query string
6	...	

```
const params = new URLSearchParams('?_page=1&_limit=10');

params.get('_page'); // 1
params.get('_limit'); // 10
params.get('_order'); // null

params.set('_page', 2);
params.toString(); // _page=2&_limit=10

params.append('_page', 3);
params.toString(); // _page=2&_limit=10&_page=3

params.has('_page'); // true
params.has('_order'); // false
```

> Src: https://developer.mozilla.org/en-US/docs/Web/API/URLSearchParams

3. Location

```
window.location;
```

	href	replace()	assign()
Type	property (get/set)	method	method
Get	current URL	N/A	N/A
Add to history list	YES	NO	YES
Delete current page from history	NO	YES	NO
When to use	get current URL	replace URL	navigate to new URL

```
// get current URL
window.location.href;

// replace current page with new URL (not able to go back)
window.location.replace('https://ezfrontend.com');

// navigate to new URL
window.location.assign('https://ezfrontend.com');
```

> Src: https://www.geeksforgeeks.org/difference-between-window-location-href-window-location-replace-and-window-location-assign-in-javascript/

4. History API

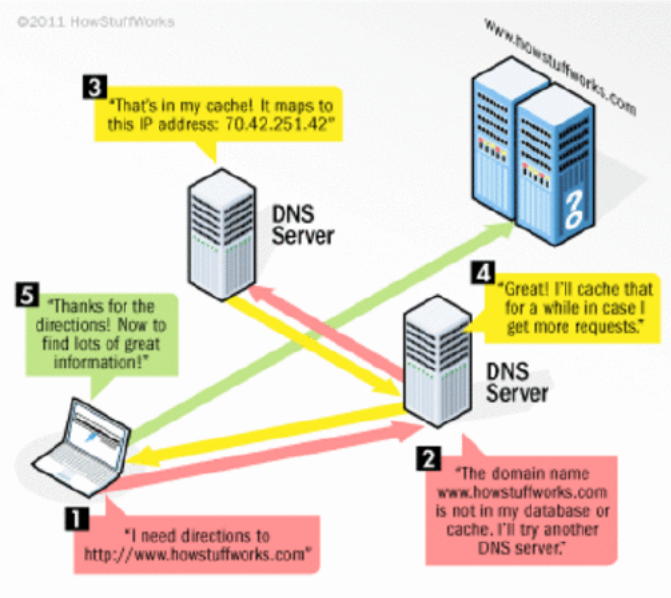
- Provide access to browser's session history
- You can navigate back and forth through the user's history

```
window.history.back();
window.history.forward();
```

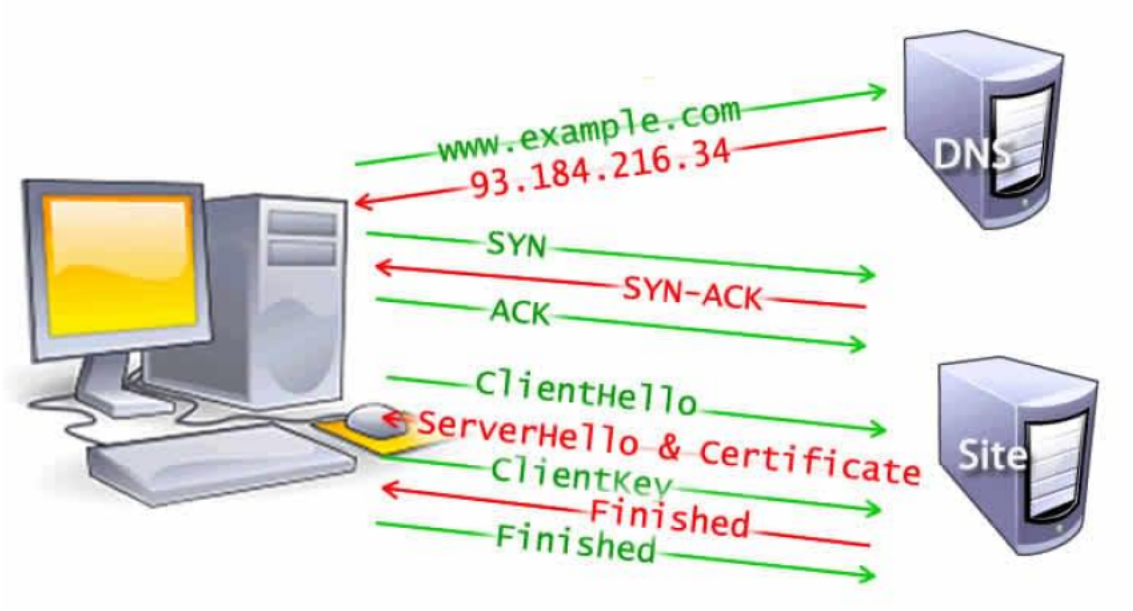
> Src: https://developer.mozilla.org/en-US/docs/Web/API/History_API

🌐 **REFERENCES**
https://www.keycdn.com/blog/difference-between-http-and-https
https://developer.mozilla.org/en-US/docs/Web/API/URL_API
https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_is_a_URL

▼ **174. What Happen When I Visit a Website?**



> Src: https://www.houkconsulting.com/2018/02/what-happens-when-visit-website/



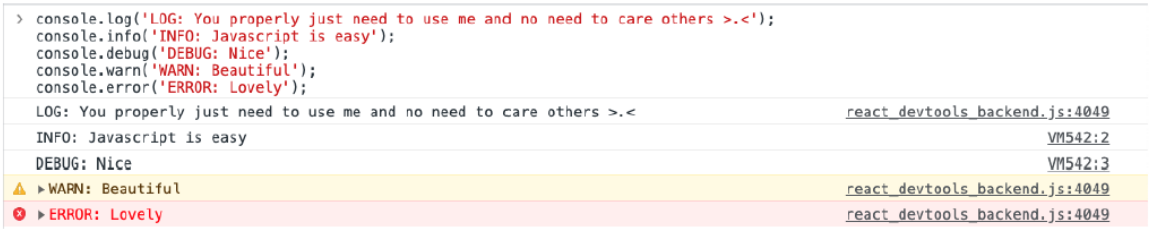
> Src: https://developer.mozilla.org/en-US/docs/Web/Performance/How_browsers_work

▼ 175. Console API

> Src: https://www.freecodecamp.org/news/javascript-console-log-example-how-to-print-to-the-console-in-js/

1. Log Variations

```
console.log('LOG: You properly just need to use me and no need to care others >.<');
console.info('INFO: Javascript is easy');
console.debug('DEBUG: Nice');
console.warn('WARN: Beautiful');
console.error('ERROR: Lovely');
```



2. Multiple Values

Group multiple values into object for better readable

```
const student = {
  id: 1,
  name: 'Alice',
  age: 18,
};

const city = {
  id: 1,
  name: 'HCMC'
};

// NOT GOOD
console.log(student);
console.log(city);

// BETTER
console.log({ student, city });
```

```
▶ {id: 1, name: 'Alice', age: 18}
▶ {id: 1, name: 'HCMC'}
```

```
▼ {student: {…}, city: {…}} i
  ▼ city:
      id: 1
      name: "HCMC"
    ▶ [[Prototype]]: Object
  ▼ student:
      age: 18
      id: 1
      name: "Alice"
    ▶ [[Prototype]]: Object
  ▶ [[Prototype]]: Object
```

Log with variables

```
const student = {
  id: 1,
  name: 'Alice',
  age: 18,
};

console.log(`My student: ${student} is lazy`);
// My student: [object Object] is lazy
```

```
const student = {
  id: 1,
  name: 'Alice',
  age: 18,
};

console.log('My student: %o is lazy', student);
```

```
My student: ▶ {id: 1, name: 'Alice', age: 18} is lazy
```

```
// OK
console.log('Name: %s and Age: %i', student.name, student.age);
// Name: Alice and Age: 18

// GOOD
console.log(`Name: ${student.name} and Age: ${student.age}`);
// Name: Alice and Age: 18
```

3. Assert

```
console.assert(condition, message);
```

- Only show message if condition is false.

```
console.assert(1 === 1, 'equal but not show in console');
console.assert(1 === 2, 'not equal and show in console');
```

```
> console.assert(1 === 1, 'equal but not show in console');
  console.assert(1 === 2, 'not equal and show in console');
▶ Assertion failed: not equal and show in console
```

4. Counting

#	Name	Desc
1	console.count(label = 'default')	count how many times that have been called
2	console.countReset(label = 'default')	reset the counter

```
console.count(); // default: 1
console.count('default'); // default: 2
console.count(); // default: 3
```

```
console.count('easy'); // easy: 1
console.count('easy'); // easy: 2

console.count('intermediate'); // intermediate: 1
console.count('intermediate'); // intermediate: 2
console.countReset('intermediate');
console.count('intermediate'); // intermediate: 1

console.count('hard'); // hard: 1
console.count('hard'); // hard: 2
console.count('hard'); // hard: 3
```

5. Track Time

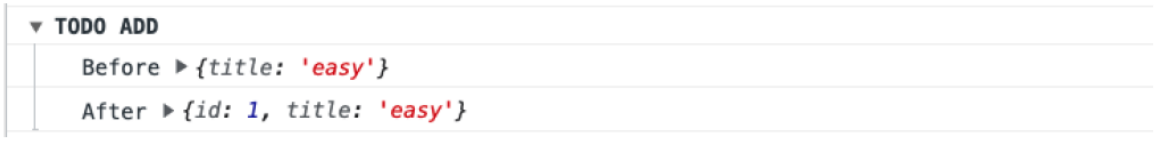
#	Name	Desc
1	console.time(name = 'default')	start timer
2	console.timeLog(name = 'default')	log time ellapsed from start
3	console.timeEnd(name = 'default')	end timer

```
console.time('loop'); // start timer

let count = 0;
for (let i = 0; i < 10000; i++) {
  count++;

  if (i % 1000 === 0) console.timeLog('loop')
}

console.timeEnd('loop'); // end timer
```

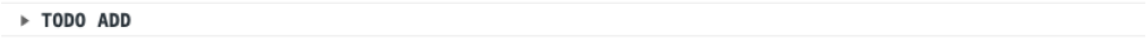
6. Groups

#	Name	Desc
1	console.group(name)	start a group with status = expanded
2	console.groupCollapsed(name)	start a group with status = collapsed
3	console.groupEnd()	end a group

```
console.group('TODO ADD');
console.log('Before', { title: 'easy'});
console.log('After', { id: 1, title: 'easy' });
console.groupEnd();
```

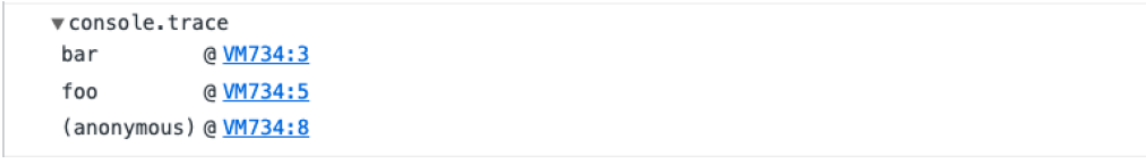



```
console.groupCollapsed('TODO ADD');
console.log('Before', { title: 'easy'});
console.log('After', { id: 1, title: 'easy' });
console.groupEnd();
```



7. Stack Trace

```
function foo() {
  function bar() {
    console.trace();
  }
  bar();
}

foo();
```



8. Table

```
console.table({
  id: 1,
  name: 'Alice',
  age: 18,
  isHero: true,
});
```

VM1876:1

(index)	Value
id	1
name	'Alice'
age	18
isHero	true

▶ Object

```
console.table([
{
  id: 1,
  name: 'Alice',
  age: 18,
  isHero: true,
},
{
  id: 2,
  name: 'Bob',
  age: 20,
  isHero: false,
},
], ['id', 'name', 'age']);
```

VM1871:1

(index)	id	name	age
0	1	'Alice'	18
1	2	'Bob'	20

▶ Array(2)

9. Styles

- use %c to start a styled text.

10. Clear

- Console.clear() to clear current console
- Using Ctrl + L to quickly clear the console

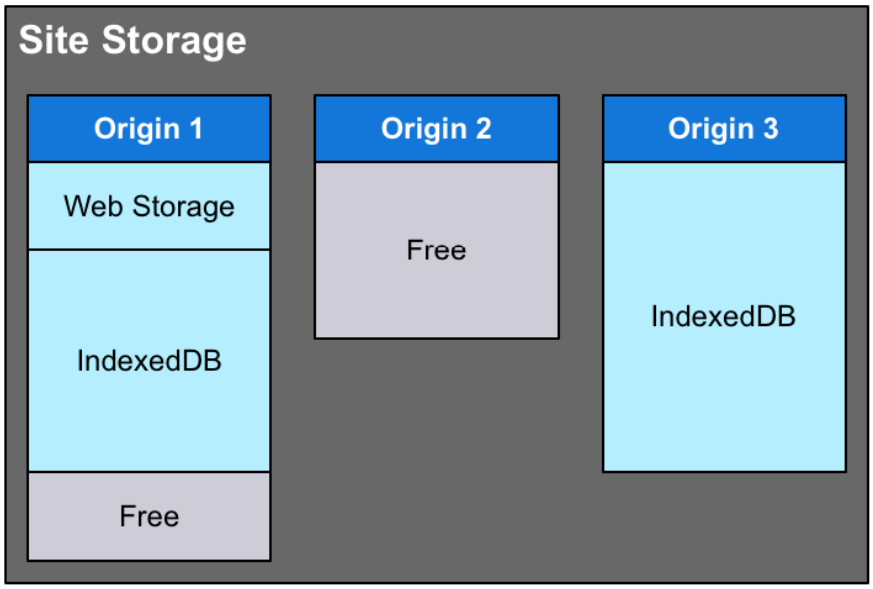
🌐 **REFERENCES**
https://developer.mozilla.org/en-US/docs/Web/API/console
https://www.freecodecamp.org/news/javascript-console-log-example-how-to-print-to-the-console-in-js/

▼ **176. Storage API**

1. Site Storage

- Site storage has many site storage unit.
- Each site storage unit represents for one origin.

- This origin can't get another origin's storage.



Site storage includes:

- localStorage
- sessionStorage
- cookies
- Cache API data
- IndexedDB database
- Service Worker registrations
- History state information
- Notification data
- ...

> Src: https://developer.mozilla.org/en-US/docs/Web/API/Storage_API

2. Storage Interface

#	Name	Desc
1	Storage.getItem(key)	Get value of **key**, return null if not exists
2	Storage.setItem(key, value)	Set value for key
3	Storage.removeItem(key)	Remove an item
4	Storage.clear()	Clear all items

3. localStorage

- The Storage object for the Document's origin
- The stored data is saved across browser sessions

- In private browsing / incognito mode, localStorage will be cleared if the private window is closed
- key/value should be a string

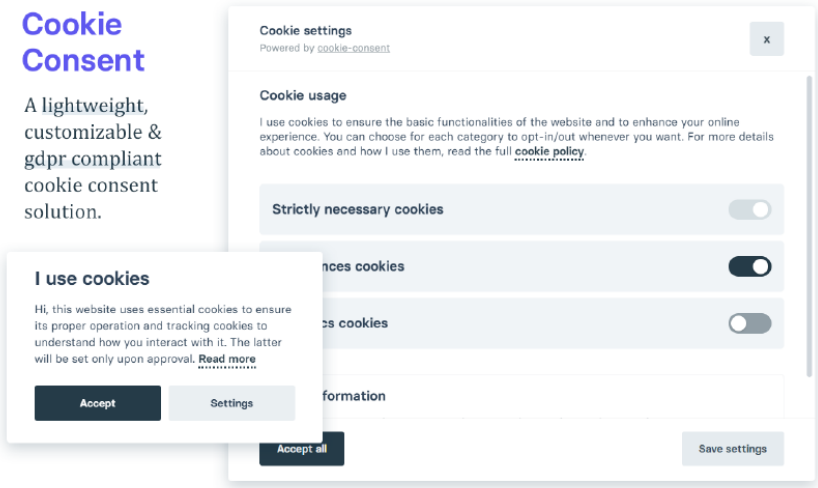
```
localStorage.setItem('name', 'Easy Frontend');
localStorage.getItem('name'); // Easy Frontend

localStorage.removeItem('name');
localStorage.getItem('name'); // null
```

4. sessionStorage

- The Storage object for the Document's origin
- It's cleared when the tab closed
- Each tab has a different sessionStorage
- Duplicating a tab -> copy tab's sessionStorage
- Data stored in sessionStorage is specific to the protocol of the page. (http and https)

```
sessionStorage.setItem('name', 'Easy Frontend');
sessionStorage.getItem('name'); // Easy Frontend

sessionStorage.removeItem('name');
sessionStorage.getItem('name'); // null
```

5. cookies



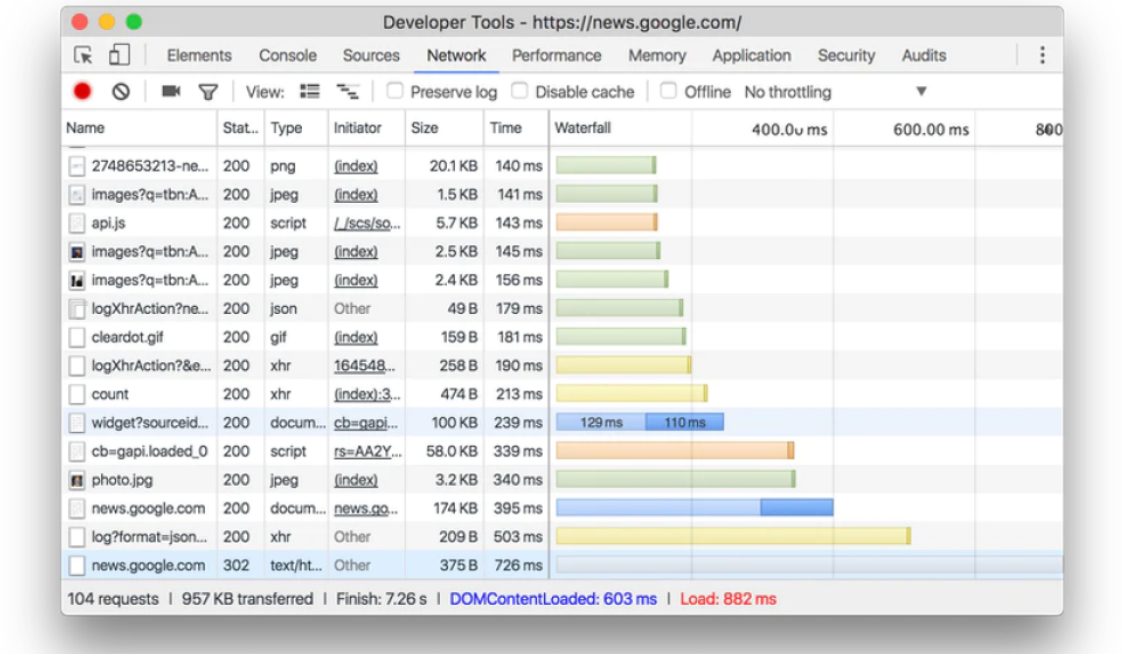
> Src: https://orestbida.com/blog/cookie-consent/

	cookies	localStorage	sessionStorage
Capacity	4kb	10mb	5mb
Accessible from	any window	any window	same tab
Expires	Manually set	Never	On tab close
Storage location	Browser and server	Browser only	Browser only
Sent with requests	Yes	No	No

> Src: https://www.freecodecamp.org/news/everything-you-need-to-know-about-cookies-for-web-development/

REFERENCES
https://web.dev/storage-for-the-web/
https://developer.mozilla.org/en-US/docs/Web/API/Storage_API
https://www.osano.com/articles/cookie-consent-requirements

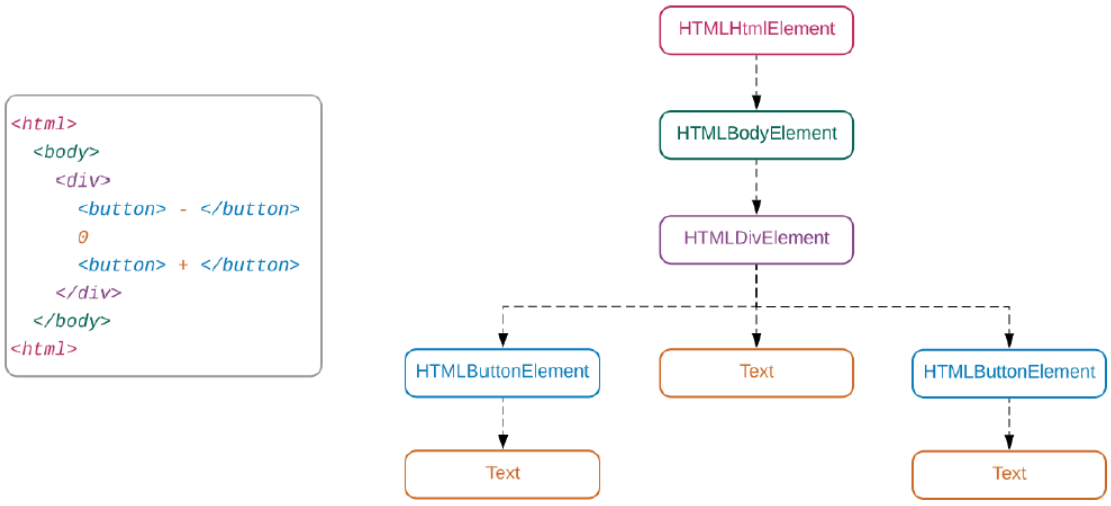
▼ **177. Browser Network**



> Src: https://developer.chrome.com/docs/devtools/network/reference

▼ SECTION 19: DOM

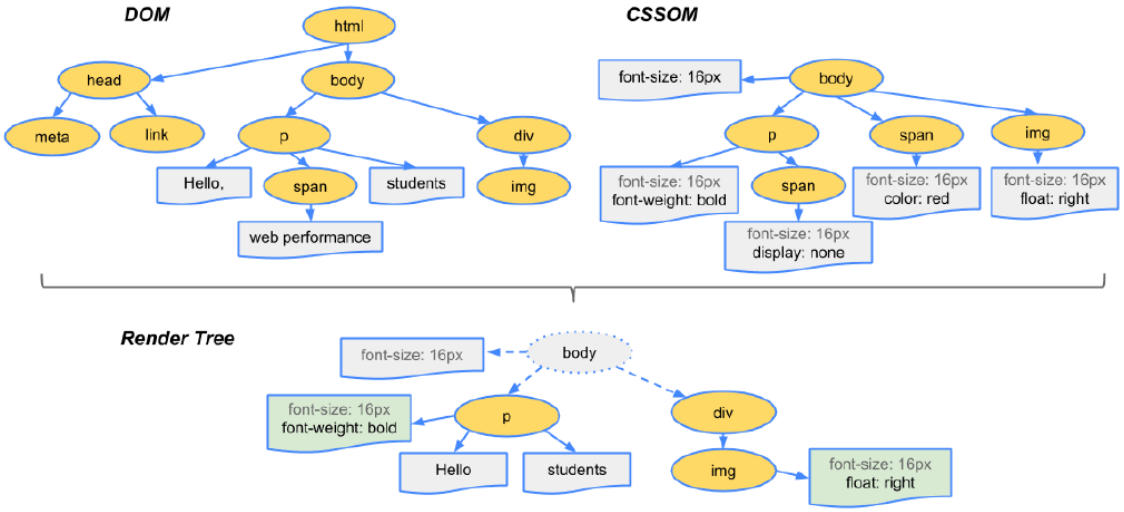
▼ **178. DOM Overview**



> Src: https://elmprogramming.com/virtual-dom.html

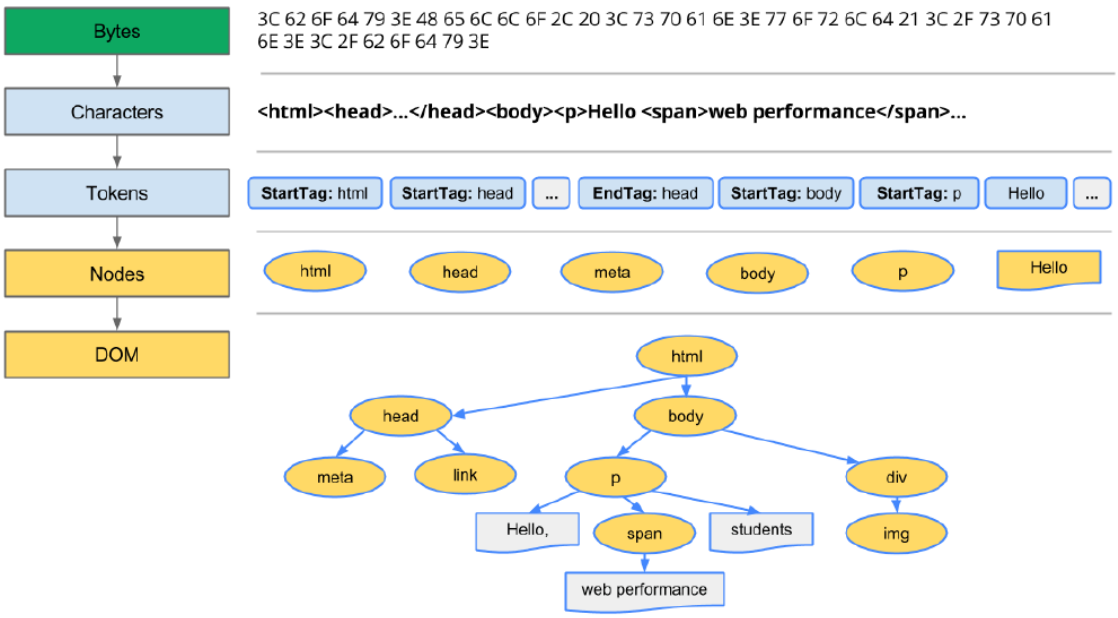
1. Redering Process Rewind

- The **DOM** and **CSSOM** trees are combined to form the **render tree**
- Render tree **contains only the nodes required to render** the page
- **Layout** computes the exact position and size of each object
- The last step is **paint**, which takes in the final render tree and renders the pixels to the screen.

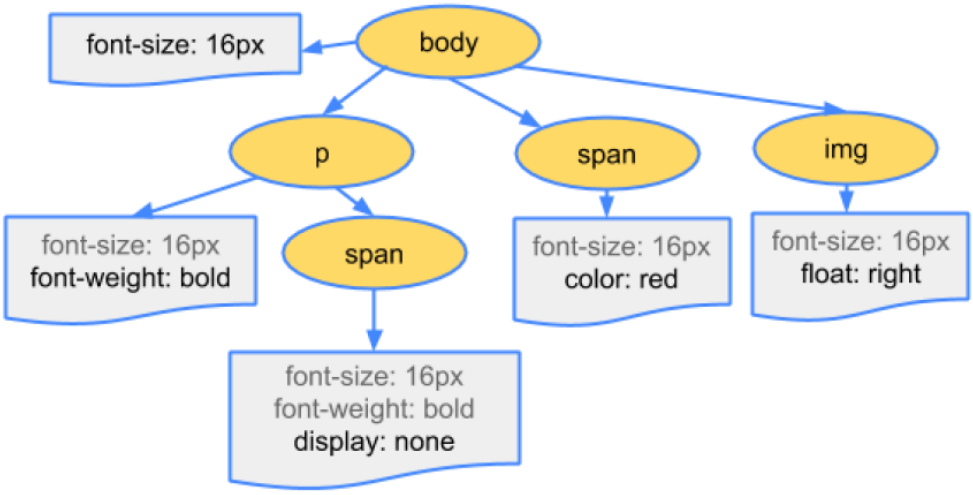


> Src: https://developers.google.com/web/fundamentals/performance/critical-rendering-path/render-tree-construction

DOM Construction

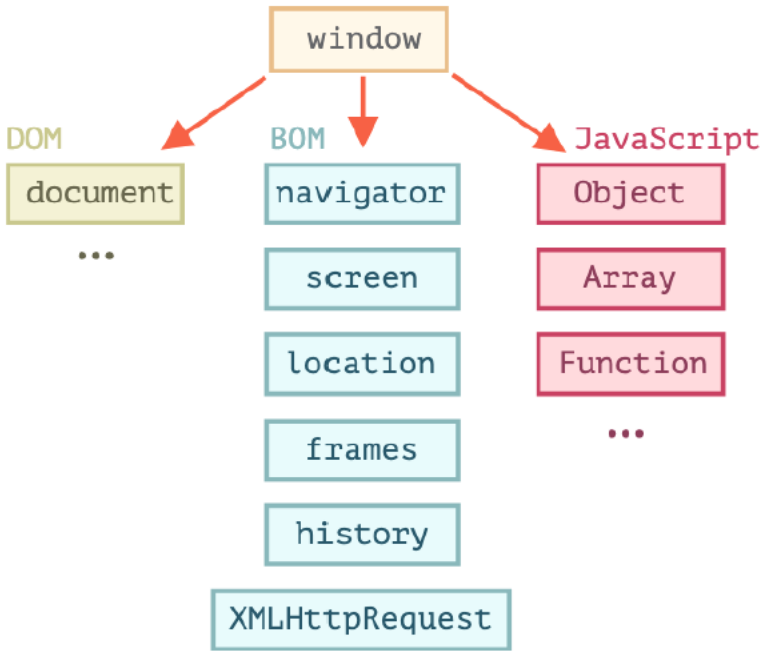


CSSOM Construction



> Src: https://developers.google.com/web/fundamentals/performance/critical-rendering-path/constructing-the-object-model

2. What is DOM?



> Src: https://javascript.info/browser-environment

#	Name	Desc
1	DOM - Document Object Model	represents and interacts with any HTML/XML document
2	BOM - Browser Object Model	represents additional objects provided by browser
3	CSSOM - CSS Object Model	represents CSS rules and sttylesheets

- DOM is **not only** for browsers
- Common properties of **navigator**:
  - navigator.userAgent: info about current browser
  - navigator.platform: info about platform

3. Things Need To Know when Work With DOM

#	Name	Desc
1	What is DOM	Tổng quan về DOM
2	DOM structure, Node, Element, ...	Cấu trúc cây DOM, mỗi Node có gì, ...
3	Query DOM	Làm sao query được một hoặc nhiều elements nào đó?
4	DOM Traversal	Duyệt phần tử trên DOM thế nào?
5	DOM Manipulation (add, update, remove)	Xử lý DOM ra sao?
6	Attach Events to DOM	Attach Events cho DOM như thế nào?
7	Also research Virtual DOM, Shadow DOM	Nên mở rộng tìm hiểu thêm về Virtual DOM và Shadow DOM

The four steps when working with DOM

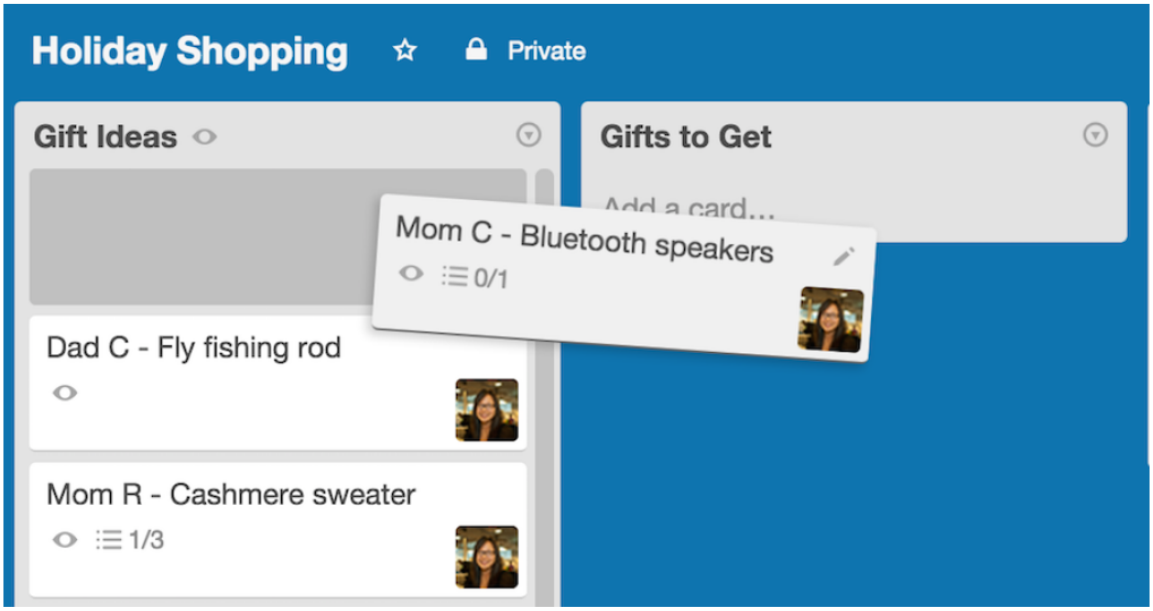
#	Name	Desc
1	Query element	Đi tìm ông thần mình muốn xử lý
2	Check element exist	PHẢI kiểm tra xem có tìm thấy không?
3	Manipulate the elemenet	Nếu tìm thấy thì xử lý ổng (update class, styles, ...)
4	Attach / Detach event listener	Attach events cho ổng, nhớ lưu ý là khi không cần nữa thì detach cái event đi

```
// 1. Query element
const title = document.getElementById('pageTitle');

// 2. Check element exist
if (title) {
  // 3. Manupulate the element
  title.textContent = 'Easy Frontend';

  // 4. Attach event listener: click
  title.addEventListener('click', () => {
    title.style.color = 'red';
  });
}
```

4. HTML DOM Example

- Access and manipulate HTML elements
- Working with form
- Drawing / Animation / ... using canvas
- Drag and drop
- ...

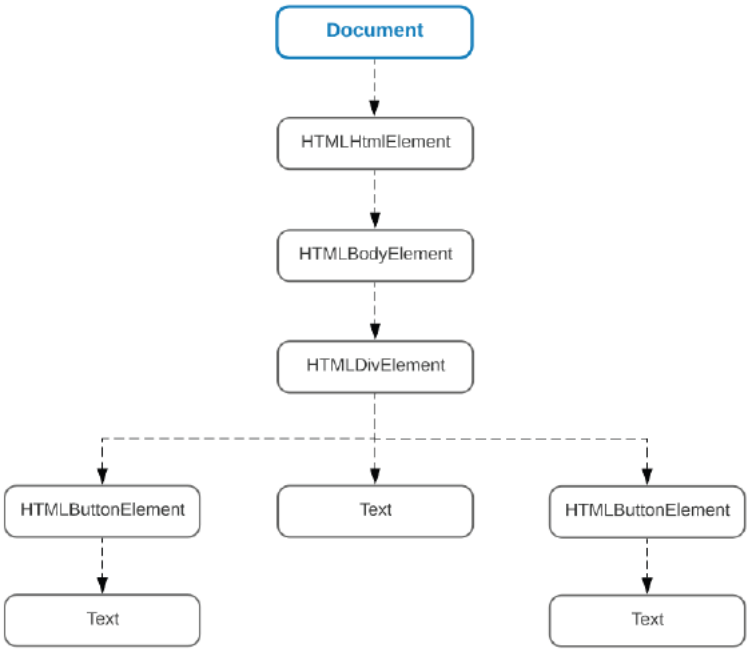
> Src: https://developer.mozilla.org/en-US/docs/Web/API/HTML_DOM_API



> Src: https://help.trello.com/article/806-moving-cards-or-lists

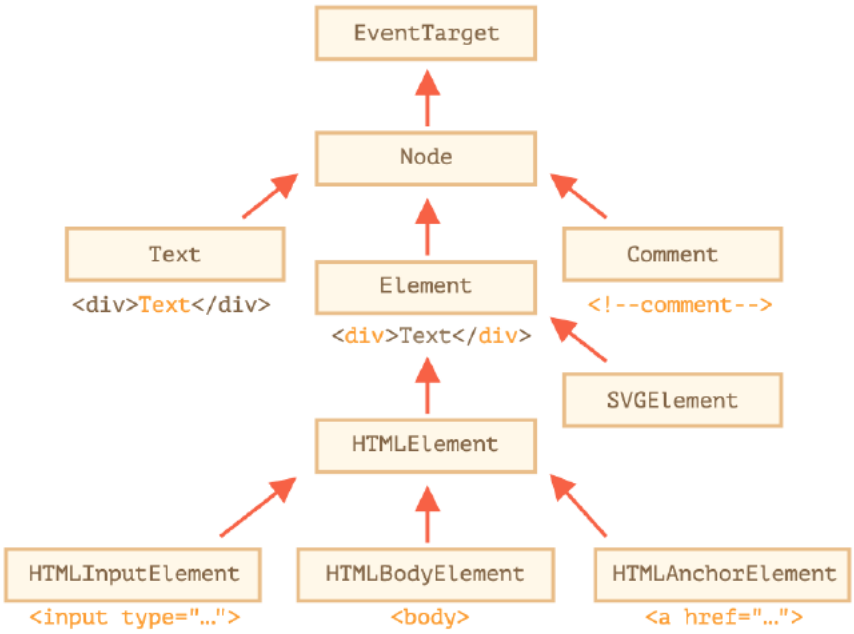
REFERENCES
https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model
https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction

▼ 179. DOM Tree Structure



> Src: https://elmprogramming.com/virtual-dom.html

1. Node Data Type



> Src: https://javascript.info/basic-dom-node-properties

- **EventTarget**: root "abstract" class
- **Node**: "abstract" class, a base for DOM nodes.

- **Element**: base class for DOM elements.
- **HTMLElement**: basic class for all HTML elements
  - HTMLInputElement - the class for `<input>` elements
  - HTMLBodyElement - the class for `<body>` elements
  - HTMLAnchorElement - the class for `<a>` elements
  - ...

> Src: https://developers.google.com/web/fundamentals/performance/critical-rendering-path/constructing-the-object-model

2. EventTarget Interface

#	Name	Desc
1	EventTarget.addEventListener()	Register an event hander of a specific event type
2	EventTarget.removeEventListener()	Remove an event listener

> Src: https://developer.mozilla.org/en-US/docs/Web/API/EventTarget

3. Node Interface

Properties

#	Name	Desc
1	Node.childNodes	return a **live** NodeList containing all children (elements, text, comments)
2	Node.firstChild	return the first direct child, otherwise return null
3	Node.lastChild	return the last direct child, otherwise return null
4	Node.previousSibling	return the previous node in the tree
5	Node.nextSibling	return the next node in the tree
6	Node.nodeName	the name of the node (BODY, DIV, #text, #document)
7	Node.**nodeType**	return an unsigned short representing the type of the node
8	Node.nodeValue	get / set value of the node
9	Node.parentNode	return the parent node
10	Node.parentElement	return an Element that is the parent of this node
11	Node.**textContent**	get / set the textual content of an element and all its desdants

Methods

#	Name	Desc
1	Node.insertBefore(newNode, referenceNode)	Insert newNode before referenceNode
2	Node.**appendChild**(childNode)	Append childNode as the last child of current node
3	Node.**cloneNode**(deep) - deep = true / false	Clone current code, deep = true -> clone the whole subtree.
4	Node.contains(otherNode)	Check if current node contain otherNode or not
5	Node.hasChildNodes()	Check if has any child node or not
6	Node.removeChild(childNode)	Remove a child node from current node
7	Node.replaceChild(newChild, oldChild)	Replace the oldChild with newChild node

> Src: https://developer.mozilla.org/en-US/docs/Web/API/Node

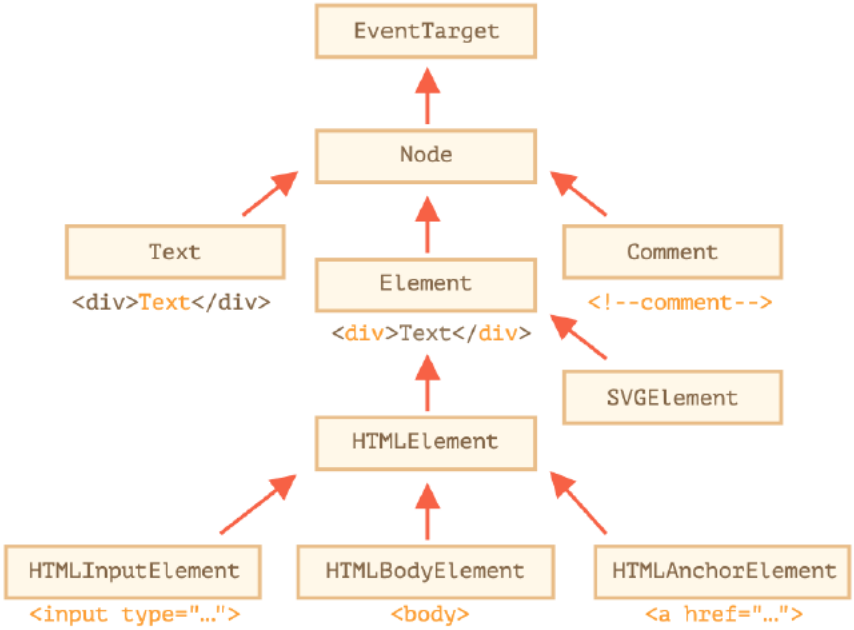
nodeType

Node.nodeType	Value	Description
Node.ELEMENT_NODE	1	An Element node like `<p>` or `<div>`
Node.ATTRIBUTE_NODE	2	An Attribute of an Element
Node.TEXT_NODE	3	The actual Text inside an Element or Attr
Node.CDATA_SECTION_NODE	4	A CDATASection, such as <!CDATA[[...]]>
Node.PROCESSING_INSTRUCTION_NODE	7	A ProcessingInstruction of an XML document, such as `<?xml-stylesheet ... ?>`
Node.COMMENT_NODE	8	A Comment node, such as `<!-- ... -->`
Node.DOCUMENT_NODE	9	A Document node
Node.DOCUMENT_TYPE_NODE	10	A DocumentType node, such as `<!DOCTYPE html>`
Node.DOCUMENT_FRAGMENT_NODE	11	A DocumentFragment node

> Src: https://developer.mozilla.org/en-US/docs/Web/API/Node/nodeType

🌐 **REFERENCES**
https://javascript.info/basic-dom-node-properties
https://developer.mozilla.org/en-US/docs/Web/API/Element

▼ **180. Document and Element Interface**



> Src: https://javascript.info/basic-dom-node-properties

1. Document Interface

Properties

#	Name	Desc
1	document.documentElement	get the html node
2	document.body	get the body node
3	document.head	get the head node
4	document.cookie (HTMLDocument)	get / set cookie
5	document.title (HTMLDocument)	get title of current document

Methods

- Query elements

#	Name	Desc
1	document.getElementById(id)	get an element by id
2	document.getElementsByClassName(name)	get a live NodeList by class name
3	document.getElementsByTagName(name)	get a live NodeList by tag name
4	document.querySelector(selector)	get the first Node that match selector
5	document.querySelectorAll(selector)	get a static NodeList that match selector

- Create

#	Name	Desc
1	document.createAttribute(name)	create return Attr node and returns it
2	document.createElement(tagName)	create the HTML element speficied by tagName and returns it

2. Element Interface

Properties

#	Name	Desc
1	Element.className	get class string
2	Element.classList	object to manage class list
3	Element.innerHTML	html markup **within** the element
4	Element.outerHTML	html markup of current element
5	Element.id	the id of the element

Element size

#	Name	Desc
1	Element.clientHeight	the inner height of the element
2	Element.clientTop	the width of the top border of the element
3	Element.clientWidth	the inner width of the element
4	Element.clientLeft	the width of the left border of the element

Scroll

#	Name	Desc
1	Element.scrollHeight	the scroll view height of an element
3	Element.scrollTop	the number of pixels the top of the document is scrolled vertically
2	Element.scrollLeft	the left scroll offset of the element
4	Element.scrollWidth	the scroll view width of the element

Traversal

#	Name	Desc
1	Element.children	return a live HTMLCollection containing all direct child elements.
2	Element.firstElementChild	returns an element's first child Element
3	Element.lastElementChild	returns an element's last child Element
4	Element.previousElementSibling	return previous element, or null if it's the first child
5	Element.nextElementSibling	return next element, or null it's the last child

Methods

- Add/Remove/Replace

#	Name	Desc
1	Element.before(...nodes)	insert a set of nodes before current element
2	Element.after(...nodes)	insert a set of nodes after current element
3	Element.prepend(...nodes)	insert a set of nodes before the first child
4	Element.append(...nodes)	insert a set of nodes after the last child
5	Element.remove()	remove current element from the tree
6	Element.replaceWith(...nodes)	replace current element with a set of nodes

- Attributes

#	Name	Desc
1	Element.getAttribute(name)	get value of an attribute by name
2	Element.hasAttribute(name)	check if an attribute is existed
3	Element.setAttribute(name, value)	set value for attribute name
4	Element.removeAttribute(name)	remove an attribute by name
5	Element.toggleAttribute(name)	toggle an attribute
6	Element.attributes()	live collection of all attribute nodes

- Query elements

#	Name	Desc
1	Element.getElementsByClassName(name)	get a live NodeList by class name
2	Element.getElementsByTagName(name)	get a live NodeList by tag name
3	Element.querySelector(selector)	get the first Node that match selector
4	Element.querySelectorAll(selector)	get a static NodeList that match selector

- Others

#	Name	Desc
1	Element.getBoundingClientRect()	return a DOMRect object providing info about the size and its position
2	Element.scroll()	scrolls the element to a particular set of coordinates
3	Element.scrollTo()	scrolls the element to a particular set of coordinates
4	Element.scrollBy()	scrolls an element by the given amount

> Src: https://developer.mozilla.org/en-US/docs/Web/API/Element

3. HTMLElement Interface

Properties

#	Name	Desc
1	HTMLElement.dataset	read/write custom data attributes (data-*)
2	HTMLElement.hidden	indicating the element is hidden or not
3	HTMLElement.innerText	the "rendered" text content of a node and its descendant
4	HTMLElement.style	representing the declarations of the element's style attribute
5	HTMLElement.title	text appears in a popup box when hover
6	HTMLElement.tabIndex	the position of the element in the tabbing order

Methods

#	Name	Desc
1	HTMLElement.blur()	remove keyboard focus
2	HTMLElement.focus()	focus to current element
3	HTMLElement.click()	send a mouse click event to the element

🌐 **REFERENCES**
https://javascript.info/basic-dom-node-properties
https://developer.mozilla.org/en-US/docs/Web/API/Element

▼ **181. Query DOM**

1. Find An Element

#	Name	Desc
1	id-named global variables	supported mainly for compatibility --> **avoid using this**
1	document.getElementById(id) (**not available on Element**)	return **null** if not found, return the element having id
2	document.querySelector(selector)	return the first Element match CSS selector, using **DFS Pre-order**, return null otherwise

```
<h1 id="title">Easy Frontend</h1>
```

```
const title = document.getElementById('title');
if (title) {
  console.log(title.textContent)
}
```

```
// remember to have hash before the id (css selector)
const title = document.querySelector('#title');
if (title) {
  console.log(title.textContent)
}
```

	getElementById	querySelector
for document only	YES	NO, also available on Element
search by	id	css selector
return	element having match id	the first node matches selector
when to use	query by id	query by others

💡 **REMEMBER: Narrow down the scope when querying**

```
<!-- ... -->

<div id="product-123" class="product">
<p class="product__name">Javascript cho người mới bắt đầu 🎉 </p>
<p class="product__price">39.99$</p>
</div>

<!-- ... -->
```

```
// DON'T
const productName =
  document.querySelector('#product-123 > .product__name');

const productPrice =
  document.querySelector('#product-123 > .product__price');
// as it need to search on the whole tree over and over again for each query
```

```
// DO: arrow down the query scope
// 1. Get the product first
// 2. From the product element, search the element inside it's subtree only
const product = document.getElementById('product-123');
if (product) {
  const productName = product.querySelector('.product__name');
  const productPrice = product.querySelector('.product__name');
}
```

2. Find All Element

#	Name	search by	Can call on element?	Live
1	**querySelectorAll** (recommended)	CSS-selector	Yes	No
2	getElementsByName	name	NO	YES
3	getElementsByTagName	tag or '*'	Yes	YES
4	getElementsByClassName	class	Yes	YES

```
<!-- ... -->

<ul id="todoList">
<li>Learn Javascript</li>
<li>Learn ReactJS</li>
<li>Learn NextJS</li>
</ul>

<!-- ... -->
```

```
// 1. Narrow down the scope of querying
const todoListElement = document.getElementById('todoList');

// 2. Make sure to check if existed before using
if (todoListElement) {
  // 3. Get list of li elements
  const todoElementList = todoListElement.querySelectorAll('li');

  // 4. Loop through it and log the content
  for (const todoElement of todoElementList) {
    console.log(todoElement.textContent);
    // Learn Javascript
    // Learn ReactJS
    // Learn NextJS
  }
}
```

Can use multiple CSS selectors by commas

```
<!-- ... -->

<div id="todoList">
  <ul class="todo-list" data-weekday="monday">
    <li>Learn Javascript</li>
    <li>Learn ReactJS</li>
    <li>Learn NextJS</li>
  </ul>

  <ul class="todo-list tuesday" data-weekday="tuesday">
    <li>Learn Javascript</li>
    <li>Learn ReactJS</li>
    <li>Learn NextJS</li>
  </ul>
</div>

<!-- ... -->
```

```
// 1. Narrow down the scope of querying
const todoListElement = document.getElementById('todoList');

// 2. Make sure to check if existed before using
if (todoListElement) {
  // 3. Some ways of query all li
```

```
  todoListElement.querySelectorAll('li');
  todoListElement.querySelectorAll('ul > li');
  todoListElement.querySelectorAll('ul.todo-list > li');

  todoListElement.querySelectorAll(
  'ul[data-weekday="monday"] > li, ul[data-weekday="tuesday"] > li'
  );
}
```

> Src: https://javascript.info/searching-elements-dom

💡 **NOTE**
- DOM larger, search too much time
- Need to narrow down scope when querying

`getElementById()`: query by id
`querySelector()`: query by css selector
`querySelectorAll()`: query all by css selector

🌐 **REFERENCES**
https://developer.mozilla.org/en-US/docs/Web/API/Document/getElementById
https://developer.mozilla.org/en-US/docs/Web/API/Document/querySelector
https://developer.mozilla.org/en-US/docs/Web/API/Document/querySelectorAll
https://javascript.info/searching-elements-dom

▼ **183. DOM Traversal**

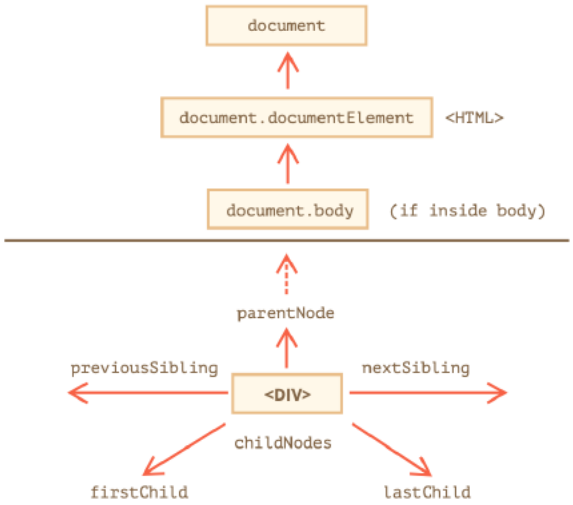
1. Overview

- **Node** can be: Text, Element, Comment, ...
- **Element**: html and svg elements like: div, p, section, h1, ...
- **Chilren**: direct child
- **Descendants**: including children, their children and so on.

```
document; // root node

document.documentElement; // html
document.body; // body
document.head; // head
```

#	Node	Element ONLY
1	Node.parentNode	Node.parentElement
2	Node.childNodes	Element.children
3	Node.firstChild	Element.firstElementChild
4	Node.lastChild	Element.lastElementChild
5	Node.previousSibling	Element.previousElementSibling
6	Node.nextSibling	Element.nextElementSibling

2. All Nodes Traversal



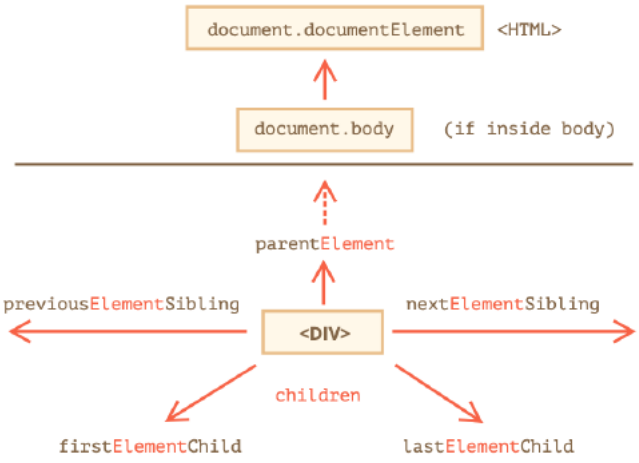
> Src: https://javascript.info/dom-navigation

```
<!DOCTYPE html>
<html>
  <head>
    <title>Parcel Sandbox</title>
    <meta charset="UTF-8" />
  </head>
  <body>
    <!-- Page Title -->
    <h1>Easy Frontend</h1>

    <p>Learning DOM is so easy! :P</p>

    <script src="src/index.js"></script>
  </body>
</html>
```

```
console.log(document.body.parentNode); // html

console.log(document.body.childNodes);
// [text, comment, text, h1, text, p, text, script, text] (9 items 🤣 )

console.log(document.body.firstChild); // text
```

```
console.log(document.body.lastChild); // text

console.log(document.body.previousSibling); // text
console.log(document.body.nextSibling); // null
```

3. Only Element Traversal



> Src: https://javascript.info/dom-navigation

```
<!DOCTYPE html>
<html>
  <head>
    <title>Parcel Sandbox</title>
    <meta charset="UTF-8" />
  </head>
  <body>
    <!-- Page Title -->
    <h1>Easy Frontend</h1>

    <p>Learning DOM is so easy! :P</p>

    <script src="src/index.js"></script>
  </body>
</html>
```

```
console.log(document.body.parentElement); // html
console.log(document.body.children);
// [h1, p, script] (only 3 items 😎 )

console.log(document.body.firstElementChild); // h1
console.log(document.body.lastElementChild); // script

console.log(document.body.previousElementSibling); // head
console.log(document.body.nextElementSibling); // null
```

4. parentNode and parentElement

- **parentNode** return the parent no matter what kind of node
- **parentElement** return the parent if it's an Element, otherwise return null

```
document.documentElement.parentNode; // document
document.documentElement.parentElement; // null
```

> Src: https://javascript.info/dom-navigation#element-only-navigation

🌐 **REFERENCES**
https://javascript.info/dom-navigation#dom-navigation-tables (Read more Nagivation on table)
https://qualitestgroup.com/insights/technical-hub/how-to-traverse-the-dom/
https://javascript.info/dom-navigation

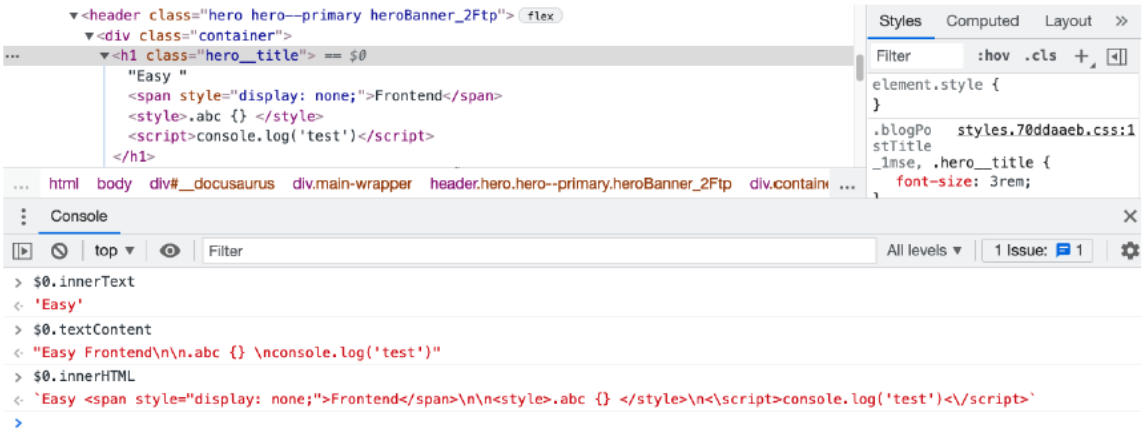
▼ 184. Create Element

```
<div id="app"></div>
```

```
const app = document.getElementById('app');
if (app) {
  // 1. Create element
  const h1 = document.createElement('h1');

  // 2. Set attributes for the new element
  h1.textContent = 'Easy Frontend';

  // 3. Add it to DOM tree
  app.appendChild(h1);
}
```

▼ 185. textContent, innerText, innerHTML

#	textContent	innerText	innerHTML
what	content of all elements	human-readable elements	inner html string
aware of styling	No	YES	No
XSS attack	No	No	YES




```
const dangerousTextFromSomewhere = "<img src='abc' onerror='alert(\"you are hacked!\")' />";
target.innerHTML = dangerousTextFromSomewhere;
```

> src: https://developer.mozilla.org/en-US/docs/Web/API/Node/textContent#differences_from_innertext

▼ **186. Update Element**

1. Element.attributes

Common attributes

#	Name	Desc
1	Element.id	the id of the element
2	Element.className	get class string
3	HTMLElement.hidden	indicating the element is hidden or not
4	HTMLElement.title	text appears in a popup box when hover
5	...	...

Element Specific Attributes

- HTMLAnchorElement `<a>`: href, rel, target, ...
- HTMLImageElement `<img>`: src, alt, ...
- ...

Custom Attributes

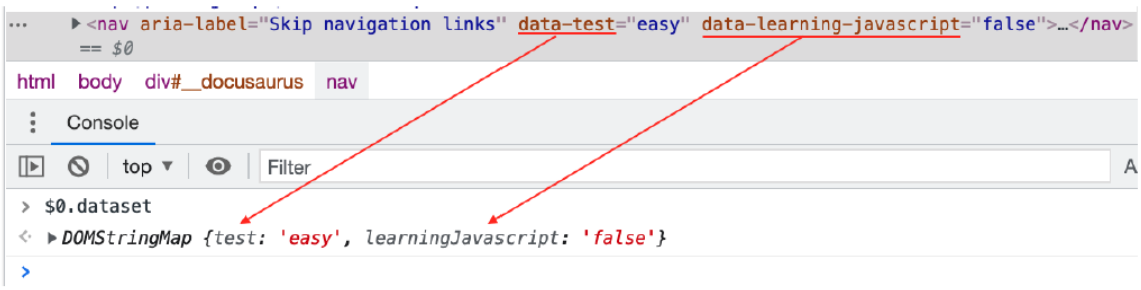
#	Name	Desc
1	Element.getAttribute(name)	get value of an attribute by name
2	Element.hasAttribute(name)	check if an attribute is existed
3	Element.setAttribute(name, value)	set value for attribute name
4	Element.removeAttribute(name)	remove an attribute by name
5	Element.toggleAttribute(name)	toggle an attribute
6	Element.attributes()	live collection of all attribute nodes

```
const title = document.getElementById('title');
  if (title) {
    title.setAttribute('learn', 'javascript');

    title.hasAttribute('learn'); // true
    title.getAttribute('learn'); // javascript
}
```

> Read more about property-attribute sync

2. HTMLElement.dataset

- HTMLElement.dataset (read-only) provide read/write access to custom data attributes (data-*)
- In HTML, **kebab-case**
- In Javascript, **camelCase**




```
<nav id="topnav" data-test="easy" data-learning-javascript="false">
<!-- ...
</nav>
```

```
const nav = document.getElementById('topnav');
if (nav) {
  // get data attributes
  nav.dataset.test; // easy
  nav.dataset.learningJavascript; // false

  // set data attribute (it will convert to string)
  nav.dataset.test = 'hard';
```

```
  nav.dataset.learningJavascript = true;
}
```

3. Element.classList

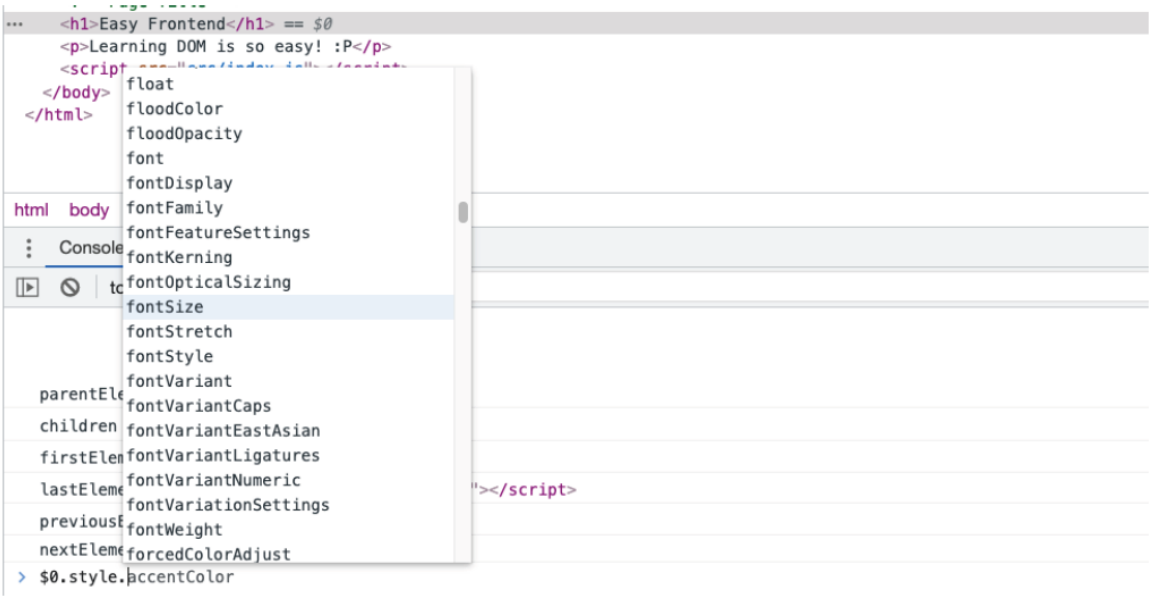
#	Name	Desc
1	Element.classList (read-only)	return a **live DOMTokenList** collection of the class
2	Element.classList.**add**(class1, class2, ...)	add class
3	Element.classList.**remove**(class1, class2, ...)	remove class
4	Element.classList.**replace**(oldClass, newClass)	replace oldClass with newClass
5	Element.classList.**toggle**(class)	toggle class

```
function handleToggleClick() {
  const accordion = document.getElementById('accordion');
  if (!accordion) return;

  accordion.classList.toggle('active');
}
```

4. HTMLElement.style

#	Name	Desc
1	HTMLElement.style.width	get / set CSS width
2	HTMLElement.style.height	get / set CSS height
3	HTMLElement.style.color	get / set CSS color
4	HTMLElement.style.**backgroundColor**	get / set CSS **background-color**
5	HTMLElement.style.fontSize	get / set CSS font-size
6	HTMLElement.style.fontWeight	get / set CSS font-weight
...	...	...

💡 **Rule**: transform **kebab-case** to **camelCase** to use as property in style object.




```
const title = document.getElementById('title');
if (title) {
  // set style
  title.style.color = 'red';
  title.style.fontSize = '24px';
  title.style.fontWeight = 'bold';

  // get style
  const computedStyle = getComputedStyle(title);
  computedStyle.color; // rgb(255, 0, 0)
}
```

REFERENCES
https://developer.mozilla.org/en-US/docs/Web/API/Element/attributes
https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement/dataset
https://developer.mozilla.org/en-US/docs/Web/API/Element/classList
https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement/style

▼ **187. Remove Element**

1. Remove Self Using Element.remove()

Auto remove message after 3 seconds

```
const message = document.getElementById('successMessage');
if (message) {
  setTimeout(() => {
    message.remove();
  }, 3000);
}
```

2. Remove All Children

#	Approach	Desc
1	`target.innerHTML = ''`	not optimized as it will trigger HTML parser
2	`target.textContent = ''` (recommended)	faster according to MDN, compared to innerHTML
3	remove every **lastElementChild**	take the code block below for ref

```
const ulElement = document.getElementById('todoList');
  if (ulElement) {
    while (ulElement.lastElementChild) {
      ulElement.removeChild(ulElement.lastElementChild);
    }
}
```

> src: https://stackoverflow.com/questions/3955229/remove-all-child-elements-of-a-dom-node-in-javascript

🌐 **REFERENCES**
https://developer.mozilla.org/en-US/docs/Web/API/Element/attributes
https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement/dataset
https://developer.mozilla.org/en-US/docs/Web/API/Element/classList
https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement/style

▼ **188. Clone Element**

- Clone node having ID can lead to duplicate element ID.
- Clone node = copies all attributes + values, including inline listeners.
- It doesn't copy event listeners added using `addEventListener()` or `node.onclick = someFunction`

```
<h1 id="title" onclick="console.log('title click')">Easy Frontend</h1>
```

Inline listeners will be copied too

```
const h1Element = document.getElementById('title');
if (h1Element) {
  // deeply clone current node and return
  const clonedH1 = h1Element.cloneNode(true);

  // change ID to avoid duplicated ID
  h1Element.id = 'newTitle';

  clonedH1.click(); // 'title click'
}
```

But it won't copy events when using addEventListeners/.onclick

```
<h1 id="title">Easy Frontend</h1>
```

```
const h1Element = document.getElementById('title');
if (h1Element) {
  h1Element.addEventListener('click', () => {
    console.log('title click');
  })

  const clonedH1 = h1Element.cloneNode(true);
  clonedH1.click(); // it doesn't log anything to console
}
```

> src: https://developer.mozilla.org/en-US/docs/Web/API/Node/cloneNode

🌐 **REFERENCES**
https://developer.mozilla.org/en-US/docs/Web/API/Element/attributes
https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement/dataset
https://developer.mozilla.org/en-US/docs/Web/API/Element/classList
https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement/style

▼ **189. Render Complex Item**

1. Issue With Complex Markup

It's ok to use **document.createElement** for **simple markup**:

```
<li>Learn Javascript</li>
```

```
function createTodoElement(todo) {
  const liElement = document.createElement('li');
  liElement.textContent = todo.title;

  return liElement;
}
```

but what if the markup is complicated like this or even more

```
<li>
  <div class="todo alert alert-secondary" role="alert">
    <div class="d-flex justify-content-between align-items-center">
      <p class="todo__title mb-0">TODO TITLE</p>

      <div class="todo__actions">
        <button type="button" class="btn btn-success mark-as-done">
        Finish
        </button>

        <button type="button" class="btn btn-danger remove">
        Remove
        </button>
      </div>
    </div>
  </div>
</li>
```

> src: https://codesandbox.io/s/js-todo-list-template-unxue?file=/index.html

2. Solution using `<template>` tag

- Using `<template>` tag
- Clone the template content
- Change the text content + attach events if needed

```
<template id="todoTemplate">
  <li>
    <div class="todo alert alert-secondary" role="alert">
      <div class="d-flex justify-content-between align-items-center">
        <p class="todo__title mb-0">TODO TITLE</p>

        <div class="todo__actions">
          <button type="button" class="btn btn-success mark-as-done">
          Finish
          </button>

          <button type="button" class="btn btn-danger remove">
          Remove
          </button>
        </div>
      </div>
    </div>
  </li>
</template>
```

```
function createTodoElement(todo) {
  const todoTemplate = document.getElementById('todoTemplate');
  if (!todoTemplate) return;

  // clone the li element
  const todoElement =
    todoTemplate.content.firstElementChild.cloneNode(true);

  // find and update title
  const todoTitleElement = todoElement.querySelector('.todo__title');
  if (todoTitleElement) todoTitleElement.textContent = todo.title;

  // TODO: attach events to buttons
  return todoElement;
}
```

🌐 **REFERENCES**
https://developer.mozilla.org/en-US/docs/Web/HTML/Element/template
https://developer.mozilla.org/en-US/docs/Web/API/DocumentFragment

▼ **190. Introduction To Browser Events**

1. Common Events

#	Event Type	Events
1	Mouse events	**click**, contextmenu, mouseover, mouseout, mousedown, mouseup, mousemove
2	Keyboard events	keydown, keyup
3	Form element events	**submit**, **focus**
4	Document events	DOMContentLoaded
5	Window events	beforeunload

> Full list events: https://developer.mozilla.org/en-US/docs/Web/Events

2. How Many Ways To Attach An Event To DOM

#	Way to attach an event	Desc
1	HTML on**event** attributes	**NOT RECOMMENDED** as it makes markup complex
2	JS on**event** properties	Only able to attach **one handler**
3	JS addEventListener	**RECOMMENDED** Able to attach **many handlers**

HTML onevent attributes / Inline event handers

```
<button onclick="alert('thank you! :P')">Click me</button>
```

JS onevent properties

- this way works but only able to attach one handler for the event.

```
<button id="clickMeButton">Click me</button>
```

```
const clickMeButton = document.getElementById('clickMeButton');
if (clickMeButton) {
  clickMeButton.onclick = function() {
    alert('thank you! :P');
  }
}
```

JS addEventListener

- **RECOMMEND** we can attach many handlers to an event.

```
<button id="clickMeButton">Click me</button>
```

```
const clickMeButton = document.getElementById('clickMeButton');
if (clickMeButton) {
  clickMeButton.addEventListener('click', function() {
    alert('thank you! :P');
  });

  clickMeButton.addEventListener('click', function() {
    alert('thank you again haha');
  });
}
```

3. Remove Event Listener If You No Longer Use

```
<button id="clickMeButton">Click me</button>
```

```
const clickMeButton = document.getElementById('clickMeButton');
if (clickMeButton) {
  function handleClick() {
    alert('thank you! :P');
  }

  clickMeButton.addEventListener('click', handleClick);

  // when no longer use, remove listener
  clickMeButton.removeEventListener('click', handleClick);
}
```

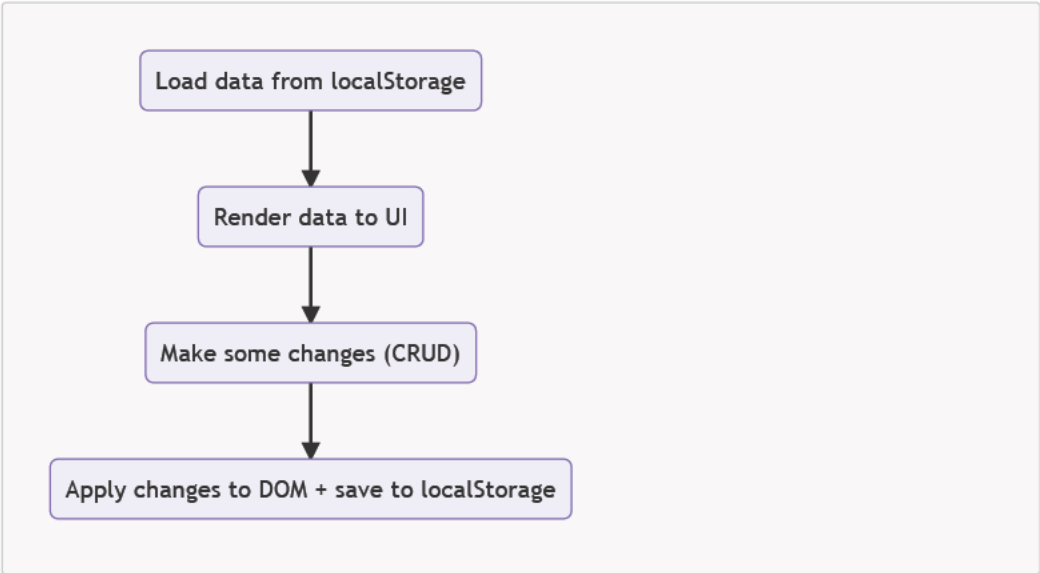
> src: https://developer.mozilla.org/en-US/docs/Web/API/EventTarget/addEventListener

🌐 **REFERENCES**
https://developer.mozilla.org/en-US/docs/Web/Events/Event_handlers
https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/Events

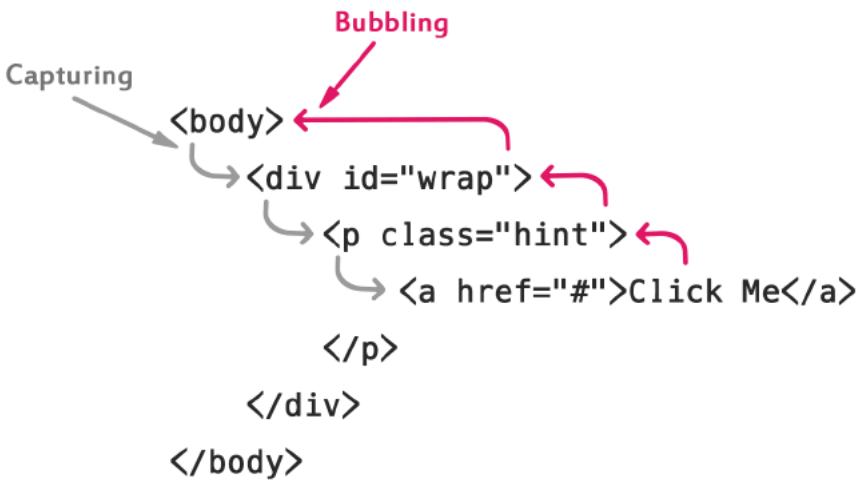
▼ **193-194. Data Persit When Reloading**

- **Issue**: data lost when reload
- **Solution**: use localStorage to store data and load from it in the next init

Data flow



▼ **195. Capturing And Bubbling**



> src: https://www.tutorialrepublic.com/javascript-tutorial/javascript-event-propagation.php

> src: https://javascript.info/bubbling-and-capturing

```
btn.addEventListener('click', function(event) => {})
```

Note about **event** object

#	Name	Desc
1	event.target	the **deepest element** that originated the event
2	event.currentTarget	the **current element** that handles the event
3	event.eventPhase	capturing=1, target=2, bubbling=3
4	event.stopPropagation()	stop bubbling up
5	event.stopImmediatePropagation()	stop bubbling up + remaining listeners attached to current element

it will log both div and button click when you click on button

```
<div onclick="console.log('div click')">
  <button onclick="console.log('button click')">click me</button>
</div>
```

Prevent event bubble to parent click using event.stopPropagation()

```
<div onclick="console.log('div click')">
  <button id="buttonId">click me</button>
</div>
```

```
const button = document.getElementById('buttonId');
if (button) {
  button.addEventListener('click', (event) => {
    event.stopPropagation();

    console.log('button click');
  });

  button.click();
  // button click
  // the click event of div is not triggered
}
```

Prevent event bubble to parent + remaining listeners using event.stopImmediatePropagation()

```
<div onclick="console.log('div click')">
  <button id="buttonId">click me</button>
</div>
```

```
const button = document.getElementById('buttonId');
if (button) {
  button.addEventListener('click', (event) => {
    event.stopImmediatePropagation();

    console.log('button click');
});

  button.addEventListener('click', (event) => {
    console.log('tada!!!');
  });

  button.click();
  // button click
  // the second click event + div click event are not triggered
}
```

🌐 **REFERENCES**
https://javascript.info/bubbling-and-capturing
https://developer.mozilla.org/en-US/docs/Web/API/EventTarget/addEventListener
https://developer.mozilla.org/en-US/docs/Web/API/Event/stopImmediatePropagation

▼ **196. Browser Default Actions**

1. Browser Default Actions

#	Event	Default action
1	mousedown	starts the selection
2	click on `<input type="checkbox" />`	check/uncheck the input
3	click on anchor tag `<a>`	redirect to the target link
4	submit - click on button/input type submit, ...	browser submit the form
5	keydown	add character into a field or other actions
6	...	...

2. How to Prevent Default Actions

#	For	How
1	HTML on**event** attributes	return false;
2	JS on**event** properties	return false;
3	JS addEventListener	event.preventDefault()

```
<!-- it won't redirect to Google when you click on this link -->
<a href="https://google.com" onclick="return false;">Go to Google</a>
```

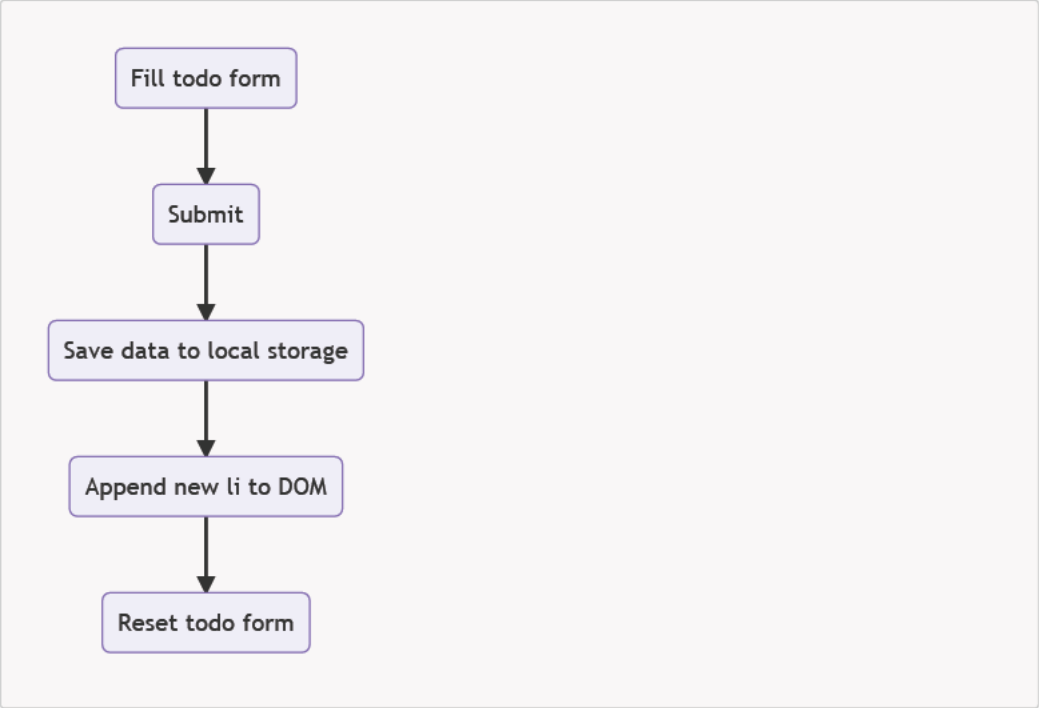
```
anchorElement.onclick = function() {
  // same as above
  return false;
}
```

```
form.addEventListener('submit', function(event) {
  // prevent form submit trigger (browser reload)
  event.preventDefault();
})
```

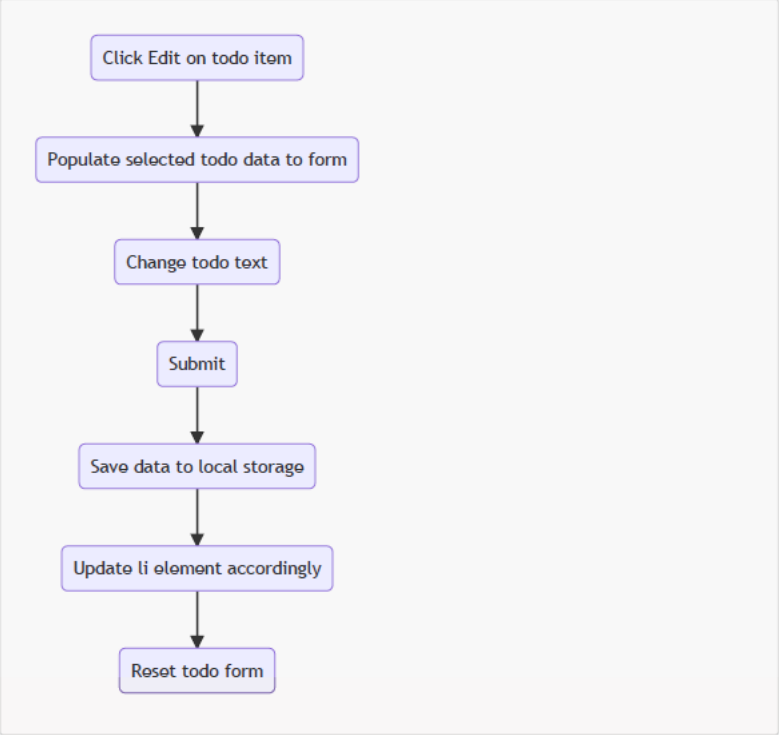
> Read about passive events: https://javascript.info/default-browser-action#the-passive-handler-option

🌐 **REFERENCES**
https://javascript.info/default-browser-action
https://developer.mozilla.org/en-US/docs/Web/API/Event/preventDefault

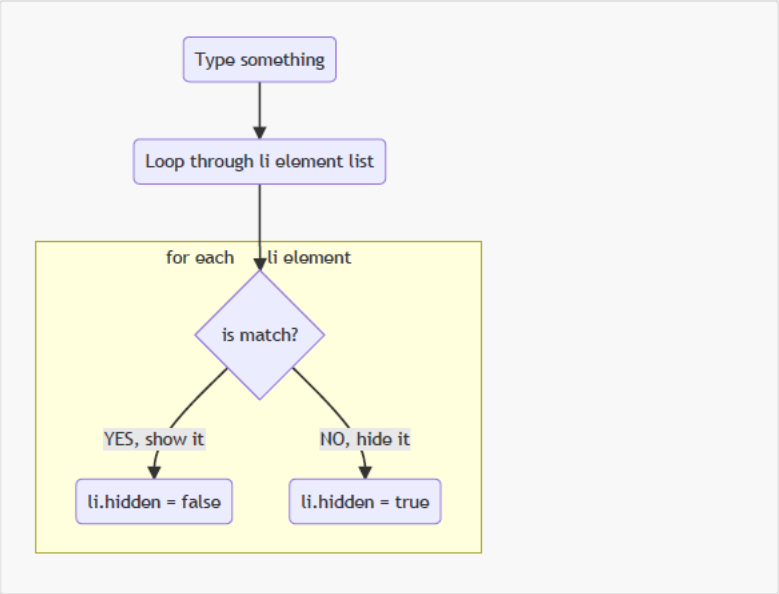
▼ **197. Create To Do**



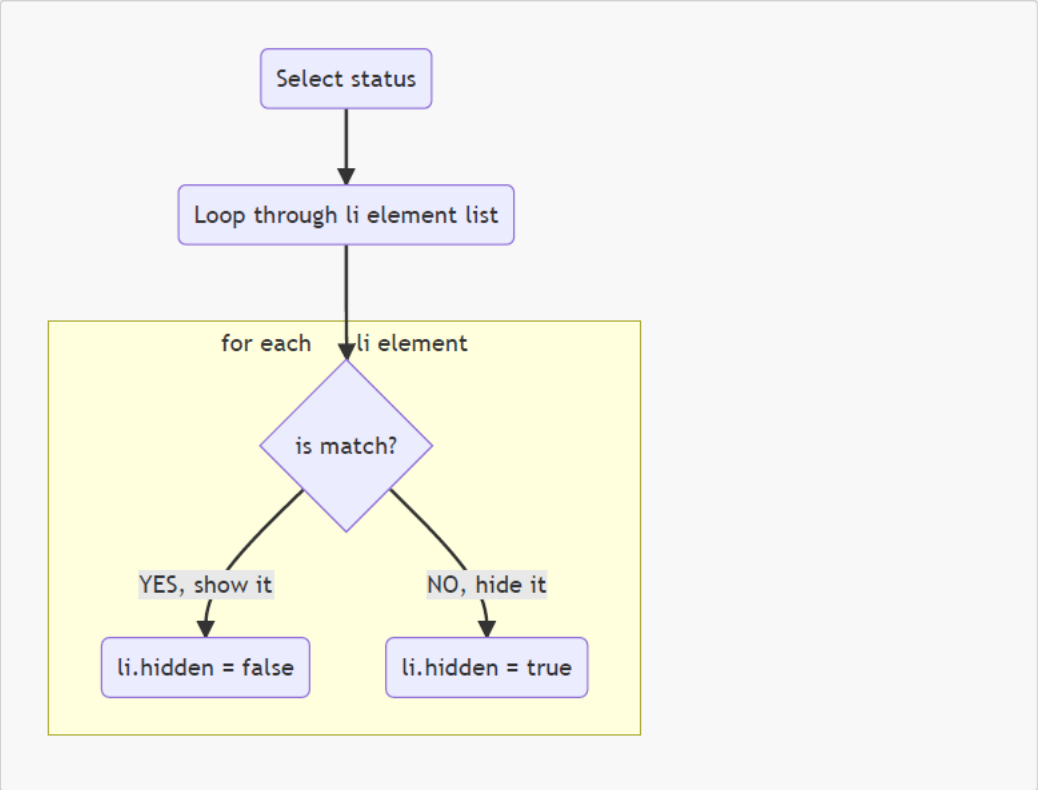
▼ 198-9. Update To Do



▼ 201. Search To Do

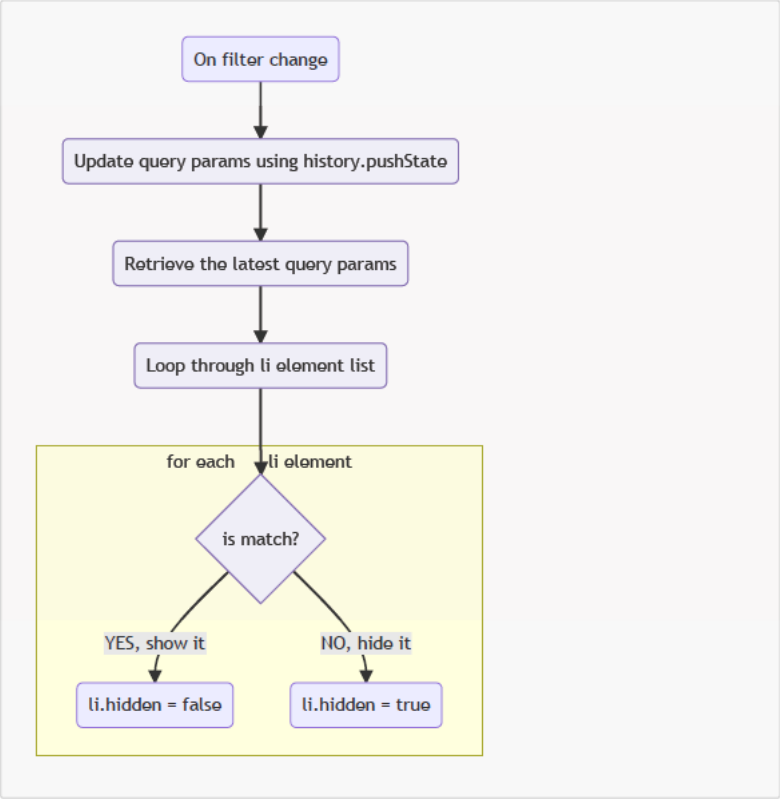


▼ **202. FilterTo Do**



▼ **204. Persist filters**

- Filters work properly
- Persist filters when reloading



▼ SECTION 22: POST UI

▼ **226. Node Package Manager**

> Src: https://waverleysoftware.com/blog/yarn-vs-npm/

1. Overview

Package manager helps to manage project's dependencies.

- Add new package
- Update a package
- Remove a package
- ...

⇒ It's a painful task, that's why need a manager for it (npm or yarn)

`package.json` file is a kind of manifest for your project.

```
{
  "name": "learn-rtx-saga",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "react-toastify": "^7.0.4",
    "redux-saga": "^1.1.3",
    "typescript": "~4.1.5",
    "yup": "^0.32.9"
  },
  "devDependencies": {
    "vite": "^2.6.0"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": "react-app"
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  }
}
```

There are two type of dependencies:

- **devDependencies**: only use in local development (built tools, dev server, ...)
- **dependencies**: it should be included in production build.

Semantic Versioning (semver)

```
major.minor.patch
1.2.3

major version = 1
minor version = 2
patch version = 3
```

Type of version

#	Name	Desc

#	Name	Desc
1	Major	Have breaking changes
2	Minor	Backward compatible new features
3	Patch	Backward compatible bug fixes

Version prefix in package.json

```
"dependencies": {
    "react": "~17.0.0", // accept all patch versions 17.0.x
    "react-hook-form": "^7.0.1" // accept all minor version
  }
}
```

> npm semver calculator: https://semver.npmjs.com/
> reference: https://docs.npmjs.com/about-semantic-versioning/

2. npm and yarn

Both **npm** and **yarn** are package manager for javascript programming language.

	npm	yarn
abbr	Node Package Manager	Yet Another Resource Negotiator
install	Auto install with NodeJS	`npm i -g yarn`
lock file	package-lock.json	yarn.lock
install package	in order	parallel (faster than npm)
recommended	--	**YES**

```
# install starter kit: create-react-app
# Option 1: using npm
npm i -g create-react-app
create-react-app my-app

# Option 2: using npx (alias of npm exec, npm v5.2+)
npx create-react-app my-app

# Option 3: using npm init (npm v6+)
npm init react-app my-app
```

```
# install starter kit: create-react-app
# Option 1: using yarn
yarn global add create-react-app
create-react-app my-app

# Option 2: using yarn create
yarn create react-app my-app
```

Common commands

	npm	yarn
Install global (can use anywhere)	npm i -g package	yarn global add package
Install packages (package.json)	npm i	yarn
Add new package for prod	npm i package	yarn add package
Add new package for dev	npm i --save-dev package	yarn add --dev package
Remove package	npm uninstall package	yarn remove package
Update package	npm update package	yarn update package
Audit	npm audit	yarn audit
Fix Audit Issues	npm audit fix	N/A

Common scripts

```
{
  "name": "learn-rtx-saga",
  "version": "0.1.0",
  "private": true,
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
}
```

	npm	yarn
dev	npm run dev	yarn dev
build	npm run build	yarn build
start	npm start	yarn start
test	npm test	yarn test
list all scripts	npm run	yarn run

🌐 **REFERENCES**
https://www.geeksforgeeks.org/difference-between-npm-and-yarn/
https://www.sitepoint.com/yarn-vs-npm/
https://classic.yarnpkg.com/en/docs/cli/
https://docs.npmjs.com/cli/v7/commands

▼ **227. Module Bundler**

> Src: https://dev.to/netlify/choosing-a-javascript-build-tool-to-config-or-not-config-2ia8

1. Overview

Module: when a js file import a another file, it's a module.

```
root
|__ utils.js # export function sayHello() {}
|__ main.js # import { sayHello } from './utils.js'
|__ index.html # <script type="module" src="./main.js" />
```

There are some common module types: CommonJS, AMD, UMD, **ESM** (ECMAScript Modules)

> **Browser support ESM:** https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Modules

Some common issues when building web application:

- Code is not optimized (minify, uglify, unused code, ...)
- New syntax is not supported on browser

⇒ **Task Runner** or **Module Bundlers**

Task Runner: Grunt, Gulp

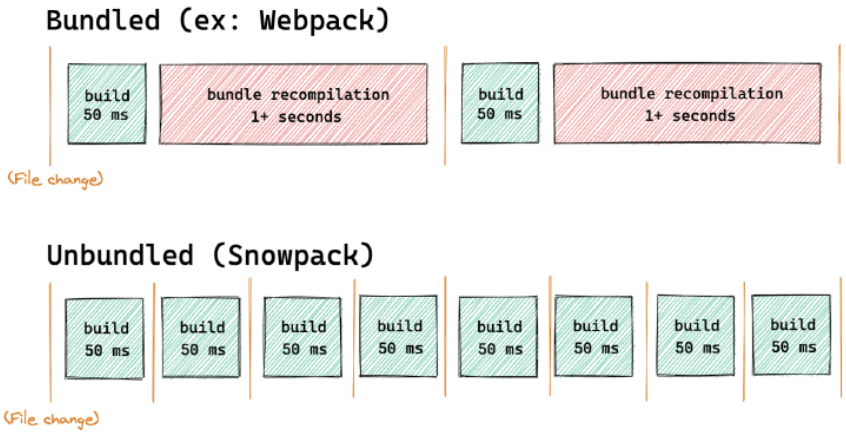
- Provides the capability to execute tasks sequentially or concurrently on the files
- No initial setup --> Define each task to have your build process.
- Less usages in projects atm.

Module Bundler: Webpack, Rollup, Parcel, ...

- Gain popular in 2015-2017
- Have abstracted the pain of tooling and provide an optimized & performance-centric solution to generate static assets.

- Code transformanation
- Tree shaking
- Hot Module Replacement
- ...

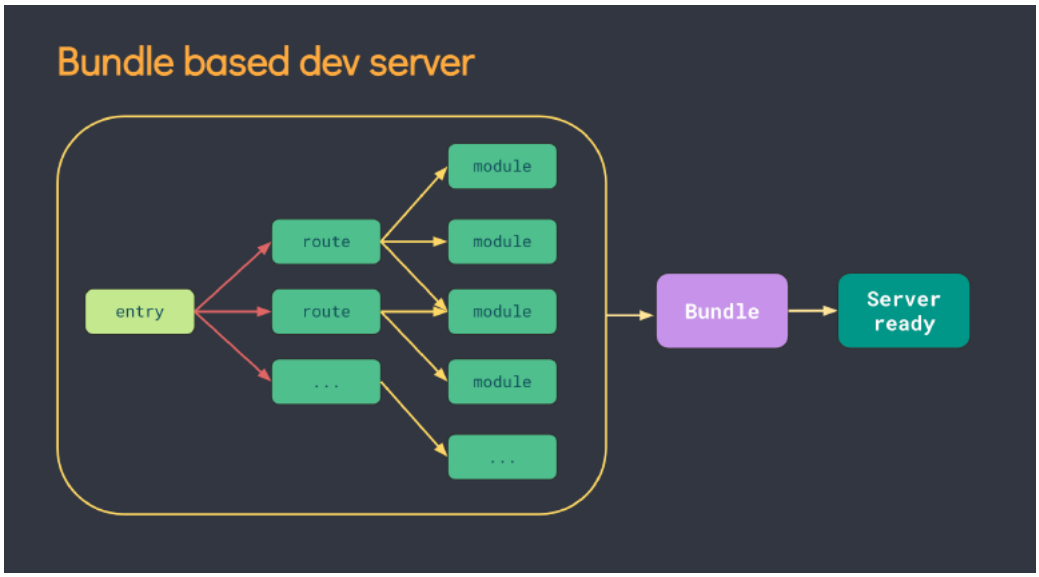
> src: https://medium.com/@rajatgms/why-do-we-need-a-module-bundler-c5ff221523f5

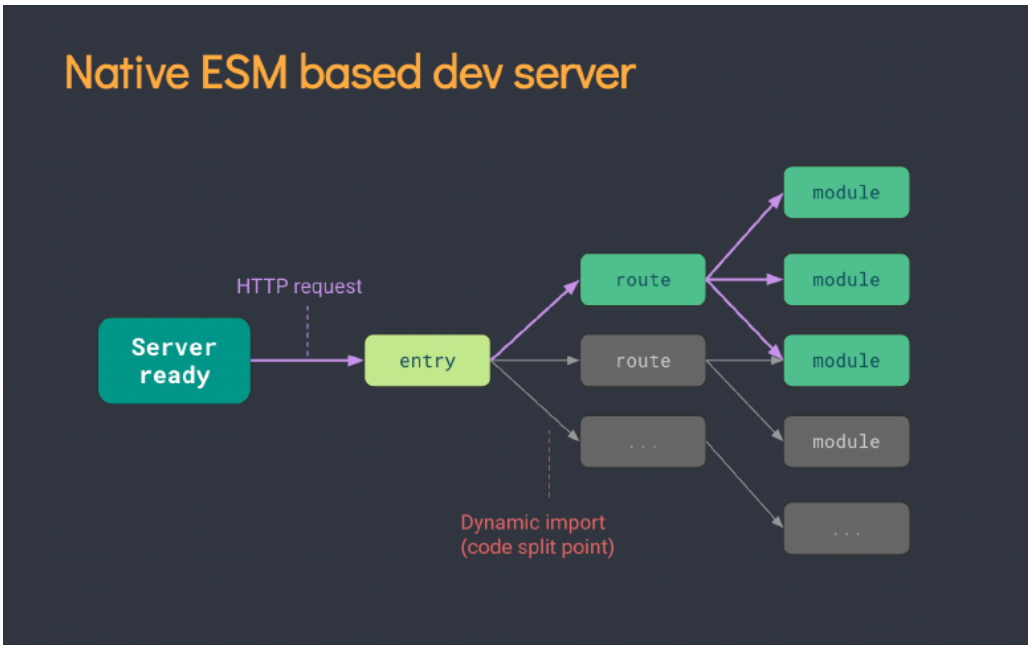


> src: https://www.snowpack.dev/concepts/how-snowpack-works

2. Vite JS

> read more: https://vitejs.dev/guide/why.html





> src: https://vitejs.dev/guide/why.html#slow-server-start

> Vite JS introduction: https://youtu.be/3ADapNcDLZU

🌐 **REFERENCES**
https://www.toptal.com/front-end/webpack-browserify-gulp-which-is-better
https://medium.com/@rajatgms/why-do-we-need-a-module-bundler-c5ff221523f5
https://webpack.js.org/concepts/#entry

▼ **228. Vite JS**

1. Run Vite JS

```
# npm 6.x
npm init vite@latest js-post-ui --template vanilla

# npm 7+, extra double-dash is needed:
npm init vite@latest js-post-ui -- --template vanilla

# yarn
yarn create vite js-post-ui --template vanilla
```

> src: https://vitejs.dev/guide/#scaffolding-your-first-vite-project

2. Folder Structure Overview

```
root
|__ dist # output folder when running build
|__ images # all images should be here
|__ js # js files
| |__ api
```

```
| |__ constants
| |__ utils
| |__ ...
|
|__ node_modules # all deps used by our project
|__ public # files to be copied to dist folder
|__ styles # css files
|__ index.html # home page (default entry point)
|__ package.json # package info
|__ vite.config.js # custom config of vitejs
|__ yarn.lock # generated by yarn
```

▼ 229. Config Multi-Page For Vite JS

By default, it only compile the index.html file. In case you want multi-page support, you need to customize `vite.config.js`

```
// vite.config.js
const { resolve } = require('path')
const { defineConfig } = require('vite')

module.exports = defineConfig({
  build: {
    rollupOptions: {
      input: {
        main: resolve(__dirname, 'index.html'),
        postDetail: resolve(__dirname, 'post-detail.html'),
        addEditPost: resolve(__dirname, 'add-edit-post.html'),
      },
    },
  },
})
```

> src: https://vitejs.dev/guide/build.html#multi-page-app

▼ 230. Public Folder And CDN Cache

1. `public` folder in ViteJS

```
root
|__ dist
|__ public # all files will be copied over to dist folder
| |__ robots.txt
| |__ favicon.ico
| |__ sitemap.xml
```

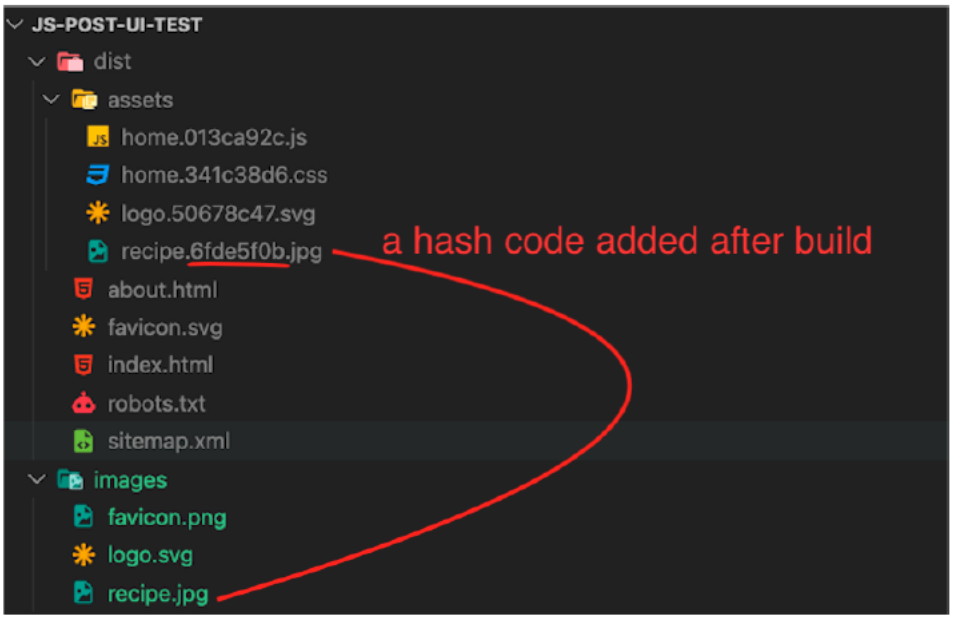
> src: https://vitejs.dev/guide/assets.html#the-public-directory

2. Adding images

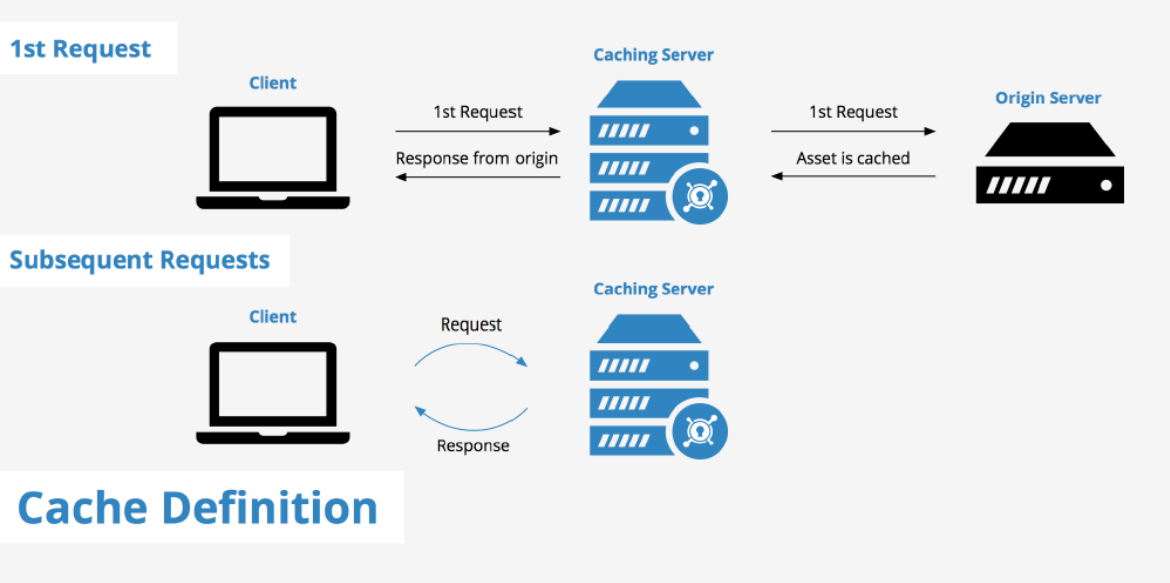
```
<img src="images/recipe.jpg" class="card-img-top" alt="recipe" />
```

```
yarn build
```

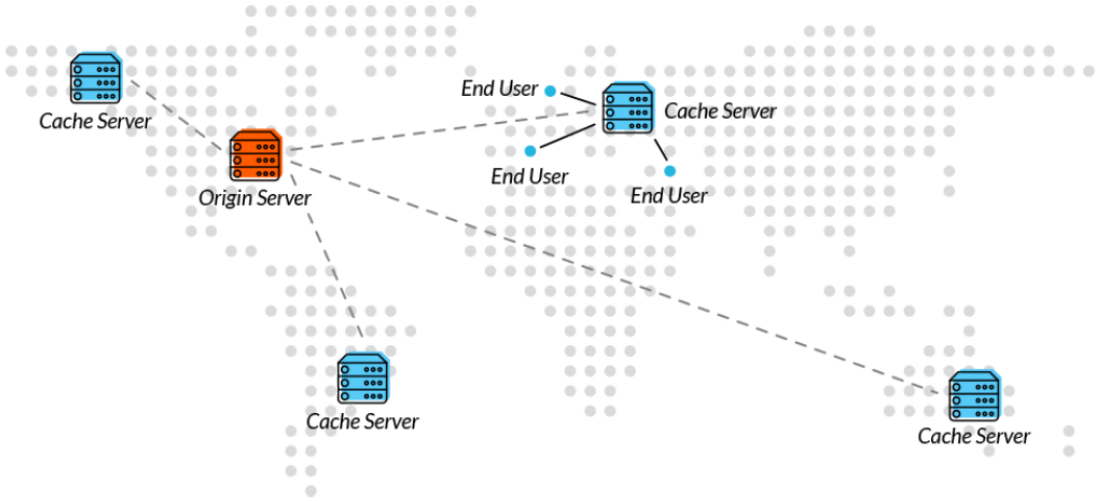
then check dist folder



3. CDN cache



> src: https://vutruso.com/bo-nho-dem-web-cache/



> src: https://awesome-tech.readthedocs.io/caching/
> read more about CDN: https://www.imperva.com/learn/performance/what-is-cdn-how-it-works/

▼ **231. From Git To Vercel**

1. Setup Git

- Install Git: https://git-scm.com/

```
# first time setup for git
# replace name and email by yours
git config --global user.name "your_name_here"
git config --global user.email your_email@gmail.com
```

```
# Init git for current folder
git init
```

- Add `.gitignore` file
  https://www.toptal.com/developers/gitignore
- Add SSH Key: https://docs.github.com/en/authentication/connecting-to-github-with-ssh/about-ssh
- Git cheatsheet: https://education.github.com/git-cheat-sheet-education.pdf

2. Push To Github

- First, go to your Github, create AN EMPTY repository
- Then add `remote` url to your local git

```
# add remote named origin
git remote add origin link_to_your_repo
```

- Create an init commit

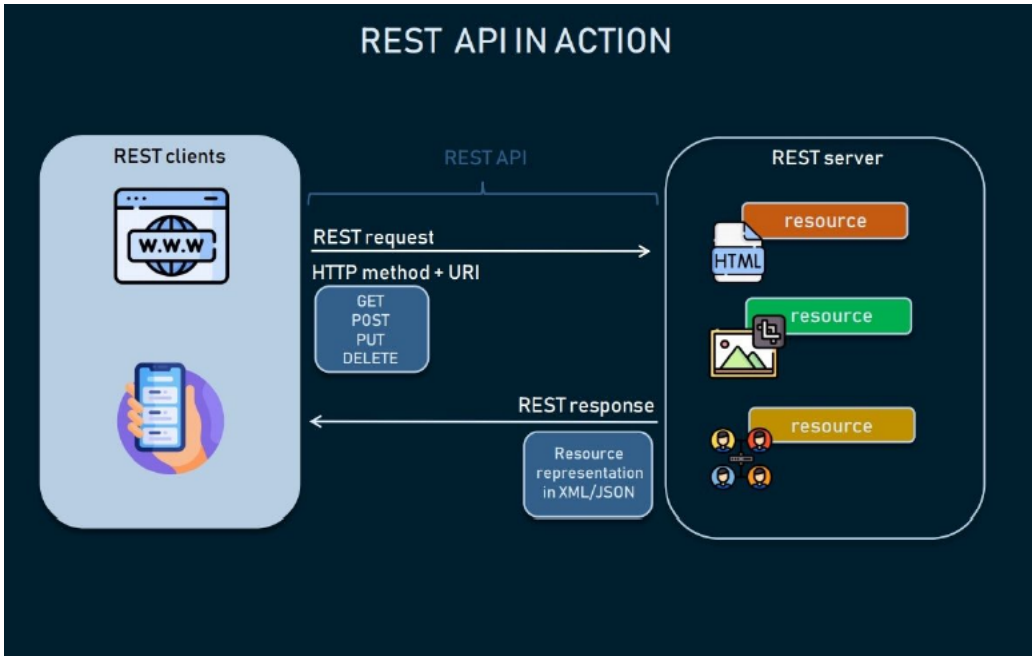
```
# stage all changes
git add .

# commit
git commit -m "Init commit"
```

```
# Switch to main branch (if not at main branch yet)
git checkout -b main

# Push code to Github
git push -u origin main
```

3. Deploy to Vercel

- Login to Vercel
- Import project from Github
- Follow guide and wait
- Tada! It's now LIVE 😊

▼ **234. API Module**



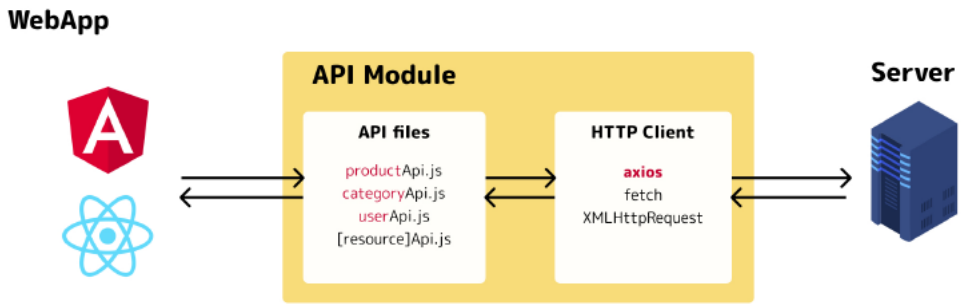
> src: https://200lab.io/blog/rest-api-la-gi-cach-thiet-ke-rest-api/

1. Why need API Module?

- Problem: When you need call API, you use `fetch()` directly. But if you want to add header for all requests? How should you do?
- It is easy to see that we need:
  - Intercept requests for setup common headers (Content-Type, Authorization, ...)
  - Intercept response for handle common errors, parse payload, ...

⇒ Need API Module to handle these problems!

2. How to organize API Module?




```
root
|__ js
| |__ api
| |__ axiosClient.js # or fetchClient.js
| |__ studentApi.js
| |__ postApi.js
| |__ `[resource]`Api.js
|
|__ index.html
|__ index.html
```

🌐 **REFERENCES**

https://developer.mozilla.org/en-US/docs/Glossary/Type_coercion
https://developer.mozilla.org/en-US/docs/Glossary/Type_Conversion
https://stackoverflow.com/questions/8857763/what-is-the-difference-between-casting-and-coercing