Tổng quan về chapter

Chương học này, không phải là cách hướng dẫn bạn học từ đầu JavaScript (TypeScript)

Mục đích: ôn tập lại các kiến thức trọng tâm & cốt lõi nhất của JavaScript (TypeScript)

Nguyên tắc: trong quá trình theo dõi và thực hành khóa học, bất cứ khi nào bạn thắc mắc/chưa hiểu về cú pháp code, bạn có thể xem lại chương học này

Các kiến thức sẽ học, bao gồm:

1. Lịch sử version của JavaScript
2. Sử dụng Var/Let/Const
3. JavaScript DataType
4. Arrow Function
5. Template String (dấu backtick)
6. Destructuring Assignment
7. Optional chaining (?.) và nullish coalescing (??)
8. ESModule và CommonJS Module
9. Async Await và Promise
10. Ôn Tập TypeScript

Lịch sử version của JavaScript

Tài liệu:
https://www.w3schools.com/js/js_versions.asp
https://en.wikipedia.org/wiki/ECMAScript_version_history
https://stackoverflow.com/a/31241842

JavaScript được tạo ra vào năm 1995, bởi Brendan Eich
Tên gọi JavaScript bị ảnh hưởng bởi Java (bởi Java rất phổ biến tại thời điểm đó)

JavaScript còn được biết tới một cái tên khác là **ECMAScript** (viết tắt là ES)

ECMAScript versions được viết tắt là: ES1, ES2, ES3, ES5, and ES6.

Từ 2016, versions được đặt tên theo năm: (ECMAScript 2016, 2017, 2018, 2019, 2020).

ES2016, ES2017… ES2020

ESNext: version tiếp theo của JavaScript

Chúng ta cần quan tâm tới version của JavaScript vì:

- Khi một version mới ra đời, nó có thể thêm các tính năng mới.
  Ví dụ ES6 (2015), bổ sung thêm **let**, **const**

- Khi code một dự án frontend/backend, cần biết version JavaScript, từ đấy có thể sử dụng cú pháp mà JavaScript hỗ trợ

Sử dụng Var/Let/Const

Nguyên tắc khi học khóa học của mình: không sử dụng var

Tài liệu:
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/var

1.Lịch sử ra đời

Var (viết tắt của từ Variable) là cách khởi tạo biến số của JavaScript (được sử dụng trong tất cả version của JavaScript)

Ví dụ:

var myName = "hỏi dân it"
Tài liệu ví dụ tại đây

Let/Const được ra đời vào năm 2015, ứng với version **ES6**

let myName = "hỏi dân it"
const myName = "hỏi dân it"

2.Nguyên tắc sử dụng

- **Không sử dụng keyword var trong quá trình bạn coding**. Thông thường bạn đọc tài liệu (ví dụ search google, stackoverflow…) có thấy keyword var, vì đấy là tài liệu cũ (trước năm 2015)

- **Bạn dùng keyword const** (viết tắt của constant) để định nghĩa biến số, mà giá trị của nó **không thay đổi** (hằng số)

- **Bạn dùng keyword let** để định nghĩa biến số, mà giá trị của nó thay đổi.

Như vậy, thứ tự ưu tiên sẽ là:

Ưu tiên sử dụng **const** (vì giá trị không thay đổi, dễ đoán & dễ fix bug)

Sau đấy tới let

Và cuối cùng (giải pháp cuối cùng, khi sử dụng const/let không dùng được), mới sử dụng tới var

JavaScript DataType

JavaScript được chia thành 2 kiểu dữ liệu:

- Kiểu dữ liệu nguyên thủy (primitive), bao gồm 7 loại:
  null
  undefined
  boolean
  number
  string
  symbol – available from ES2015
  bigint – available from ES2020

- Kiểu dữ liệu tham chiếu (Reference) : object, array

Các kiểu dữ liệu được sử dụng phổ biến nhất:

1. String (chuỗi)

```
const name = “Hỏi Dân IT”
```

2. Number (số): không phân biệt số nguyên, số dương, số thực, số phức…

```
const age = 25
const score = 1.5
```

3. Object/Array

```
const person = {
        name: “Hỏi Dân IT”,
        age: 25
}

const country = [ “VietNam”, “Lao”, “Campuchia”]
```

Arrow Function

Tài liệu:
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions

Tên gọi: Arrow Function (hàm mũi tên), vì có hình thù giống mũi tên

Ví dụ:

const sum = (a, b) => a + b

Lý do sử dụng arrow function ?
Sử dụng arrow function khác gì so với function truyền thống ?

Về bản chất (các tính năng cơ bản), arrow function có chức năng giống hệt cách định nghĩa function truyền thống, không có sự khác biệt.

Điều này có nghĩa là, **bạn có thể sử dụng arrow function/function truyền thống như nhau**, miễn sao bạn cảm thấy thoải mái

Arrow function chỉ khác function truyền thống khi bạn dùng trong OOP (ví dụ sử dụng toán tử this, constructor…)

Lý do cá nhân mình (với yêu cầu cơ bản), sử dụng arrow function: vì code ngắn và developer (dev) thường lười. Cách nào ngắn, tiện lợi thì ưu tiên sử dụng.

Template literals (Template strings) | Backtick

Tài liệu:
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Template_literals

Khi sử dụng với string, chúng ta hay dùng dấu nháy đơn hoặc dấu nháy đôi để nối chuỗi.

Dấu backtick (nháy chéo - nằm dưới phím ESC trên keyboard) cũng giải quyết vấn đề trên, tuy nhiên tiện lợi hơn.

`string text ${expression} string text`

Mục đích sử dụng dấu nháy chéo (backtick) là giúp code của bạn dễ đọc, dễ hiểu và dễ sử dụng

Destructuring Assignment

Tài liệu:
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment

Sử dụng với Object là chủ yếu:
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment#object_destructuring

```
const user = {
  id: 42,
  isVerified: true,
};

const { id, isVerified } = user;

console.log(id); // 42
console.log(isVerified); // true
```

Sử dụng với Array:
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment#array_destructuring

```
const foo = ["one", "two", "three"];

const [red, yellow, green] = foo;
console.log(red); // "one"
console.log(yellow); // "two"
console.log(green); // "three"
```

Toán tử ?. và ??
Tài liệu:
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Optional_chaining

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

```
const adventurer = {
  name: 'Alice',
  cat: {
    name: 'Dinah',
  },
};

const dogName = adventurer.dog?.name;
console.log(dogName);
// Expected output: undefined

console.log(adventurer.someNonExistentMethod?.());
// Expected output: undefined
```

Lưu ý: chỉ sử dụng khi bạn hiểu rõ bạn đang làm gì, và hạn chế tối đa việc lạm dụng nó (vì có nó che dấu bug)

ES Module và CommonJS Module
Tham khảo: https://github.com/wooorm/npm-esm-vs-cjs

1.Khái niệm

ES Module (ESM):

- Được giới thiệu trong ECMAScript 2015 (ES6).
- Được hỗ trợ từ các version Node.js 12 trở đi
- Dùng cú pháp **import** và **export**.

CommonJS (CJS):

- Là hệ thống module mặc định của Node.js từ khi ra đời.
- Dùng cú pháp **require** và **module.exports**.

2. Khi nào sử dụng
ESM:
- Khi làm việc với các trình duyệt (browser) hiện đại. Ví dụ như bạn code frontend react/angular/vue
- Khi cần hỗ trợ async module loading.
- Tương lai của JavaScript module.

CJS:

- Khi làm việc với các dự án Node.js, mà không cấu hình hỗ trợ ESM
- Khi cần sử dụng nhiều package chưa hỗ trợ ESM.

Trong khóa học, mình sẽ sử dụng cú pháp của CommonJS (require/module.exports), vì không cần cấu hình gì thêm, node.js đã hỗ trợ sẵn

3. Các khái niệm liên quan

Khái niệm javascript modules, tham khảo tại đây
Với Node.js, mỗi file được xem là một module

Khái niệm import, tham khảo tại đây
Khái niệm require/module.exports tham khảo tại đây

async await promise

Asynchronous cho phép Node.js to thực thi tasks (như là reading files, querying databases, making HTTP requests) mà không làm ảnh hưởng/ngăn cản việc thực thi của các khối code khác (blocking the execution of other code)

Các cú pháp xử lý bất đồng bộ với javascript:

- Cú pháp sử dụng **callback**
- Cú pháp **Promise**: ra đời vào năm 2015 (ES6)
- Cú pháp **Async/Await** : ra đời năm 2017 (ES2017)

Sử dụng với TypeScript

Nếu bạn chưa biết gì về TypeScript, học nhanh cú pháp tại đây

Nguyên tắc:

TypeScript = JavaScript + khai báo Type

Nếu bạn code & hiểu được code JavaScript, chắc chắn bạn sẽ code được TypeScript

Mục đích sử dụng TypeScript trong khóa học:

- Coding được gợi ý code
- Coding phát hiện lỗi (bug) dễ hơn