C4E JS - Student book

I. Async/Await là gì?

1. Giới thiệu cơ chế bất đồng bộ (async)

- Trước khi đi tìm hiểu về một trong những khái niệm tương đối phức tạp này, chúng ta cần phải nắm được cơ chế hoạt động **bất đồng bộ (async)** của JavaScript.
- **Bất đồng bộ (Async)** là đặc thù của JS, bạn có thể hiểu đơn giản là JS chạy code **không chờ đợi**. Ví dụ sau đây sẽ giúp bạn hiểu hơn về cơ chế này:

```
console.log("Hello my name is ONE");
console.log("Hello my name is TWO");
console.log("Hello my name is THREE");
```

- Chạy đoạn code trên bạn sẽ thấy code được thực hiện lần lượt từ trên xuống dưới:

```
Hello my name is ONE
Hello my name is TWO
Hello my name is THREE
```

- Sửa lại ví dụ trên một chút như sau:

```
console.log("Hello my name is ONE");

setTimeout(function(){
    console.log("Hello my name is TWO");
}, 3000);

console.log("Hello my name is THREE");
```

- Phần code bao ngoài câu lệnh **console.log("Hello my name is TWO");** bạn có thể tạm hiểu như sau: *ra lệnh cho JS **sau 3000 milisecond (tương đương 3 second) thì mới được in ra đoạn text "Hello my name is TWO"***. (Tham khảo thêm: https://www.w3schools.com/jsref/met_win_settimeout.asp)
- Kết quả:

```
Hello my name is ONE
Hello my name is THREE
Hello my name is TWO
```

→ Bạn hãy để ý đến dòng lệnh console.log() cuối cùng. Thay vì phải chờ đợi 3 giây để **"Hello my name is TWO"** được in ra màn hình rồi nó mới chạy, thì nó đã chạy ngay và cho ra kết quả trước khi **"Hello my name is TWO"** được in ra. Lý do là vì thời gian thực hiện của nó nhanh hơn. Cơ chế hoạt động này được gọi là **bất đồng bộ**

→ Cơ chế **bất đồng bộ** tóm gọn lại như sau:

- Tất cả các dòng lệnh, chức năng được chạy đồng thời. Dòng lệnh, chức năng nào nhanh hơn thì sẽ thực thi xong trước.
- Nếu 2 dòng lệnh, chức năng cùng tốc độ, thì sẽ ưu tiên theo thứ tự từ trên xuống dưới.

Nhận xét: JavaScript hoạt động với cơ chế bất đồng bộ như trên, vậy làm thế nào để quản lý tốt kết quả của một hành động bất đồng bộ (Async)? **Async/Await** sẽ giúp chúng ta làm điều này.

2. Khái niệm

- Việc sử dụng **Async/Await** giúp chúng ta viết code trông có vẻ đồng bộ (synchonous - chạy code tuần tự từ trên xuống dưới), nhưng thật ra lại chạy bất đồng bộ (asynchonous).

3. Cú pháp, cách sử dụng

- Trước hết, chúng ta sẽ khai báo một function, function này nhận vào một số, đơn vị là miligiay (1000ms = 1s). Sau đó function này sẽ đợi thời gian trôi qua bằng đúng thời gian nhận vào.
- Function này hoạt động dựa trên function **setTimeout()** đã được đề cập đến ở phần 1. Bạn có thể tìm hiểu kỹ hơn về **function setTimeout()** tại đây.

```
function wait(ms) {
    return new Promise(r => setTimeout(r, ms))
}
```

- Nếu bạn đang băn khoăn về từ khóa **Promise**, thì về cơ bản **async/await** được xây dựng dựa trên **Promise**. Promise là một khái niệm tương đối phức tạp khi mới làm quen. Nhưng đừng lo lắng, bạn có thể tạm thời gạt **Promise** sang một bên, hãy chỉ quan tâm đến chức năng của **function wait** thôi. Lý do đơn giản là vì **function wait** sẽ chỉ đóng vai trò hỗ trợ chúng ta trong quá trình tìm hiểu **async/await** thôi.

  Bạn có thể tìm hiểu về Promise ở đây.

- Vì **async/await** được sử dụng với function, nên chúng ta sẽ viết **function main** để chứa luồng chạy chính của code, đồng thời gọi luôn **function main**:

```
function wait(ms) {
    return new Promise(r => setTimeout(r, ms))
}

function main() {
    console.log('ONE');
};

main();
```

- Bây giờ chúng ta sẽ thêm 2 dòng lệnh sau ngay phía dưới **console.log('ONE');**:

```
function wait(ms) {
    return new Promise(r => setTimeout(r, ms))
}

function main() {
    console.log('ONE');
    wait(2000);
    console.log('TWO');
};

main();
```

- Chạy thử đoạn code trên, bạn sẽ thấy `ONE` và `TWO` được in ra gần như đồng thời, mặc dù ở giữa 2 câu lệnh `console.log()` chúng ta đã chèn thêm **function wait**
- Khi chèn thêm **function wait** với parameter là 2000, tương đương 2 giây, chúng ta đang mong muốn câu lệnh `console.log('ONE');` thực hiện trước, 2 giây sau thì câu lệnh `console.log('TWO');` mới được thực hiện.
- Để làm được điều này, chúng ta thêm:
    - từ khóa **async** vào trước từ khóa **function** của **function main**.
    - từ khóa **await** vào trước câu lệnh mà chúng ta muốn nó phải thực hiện xong thì các câu lệnh dưới nó mới được thực hiện.

```
function wait(ms) {
    return new Promise(r => setTimeout(r, ms))
}

async function main() {
    console.log('ONE');
    await wait(2000);
    console.log('TWO');
};

main();
```

- Chạy thử đoạn lệnh trên, bạn sẽ thấy thực sự là `"ONE"` đã được in ra màn hình, 2 giây sau `"TWO"` mới được in ra màn hình. Lý do là vì đằng trước **function wait** đã có từ khóa **await**, nó đảm bảo tất cả các câu lệnh phía dưới nó đều phải đợi cho đến khi **function wait** được chạy xong.

Lưu ý:

- Function phải có **async** thì mới có thể sử dụng được **await**.
- Tùy trường hợp mới áp dụng **async/await**, không nên lạm dụng **async/await**, quá lạm dụng sẽ khiến cho chương trình của bạn chạy chậm đi, vì mỗi lần sử dụng **async/await**, bạn sẽ cần phải chờ cho xử lý của **await** kết thúc.

II. TỔNG KẾT

- Trong các chương tiếp theo, chúng ta sẽ thực sự bước chân vào thế giới web.
- **Async/Await** sẽ áp dụng rất nhiều khi chúng ta làm việc với dữ liệu và mạng để hiển thị và bố trí sắp xếp dữ liệu lên web.

Bài tiếp theo HTML