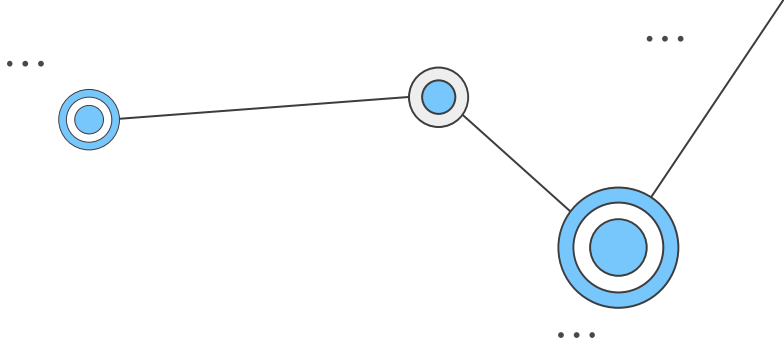
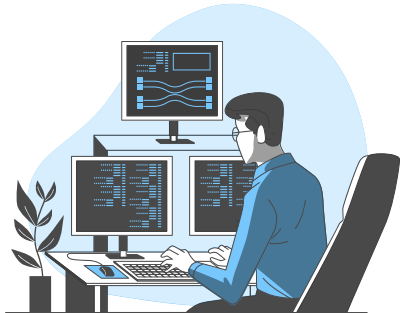
JS



Khóa học Backend

Bài 07: Javascript cơ bản (Tiết 3)



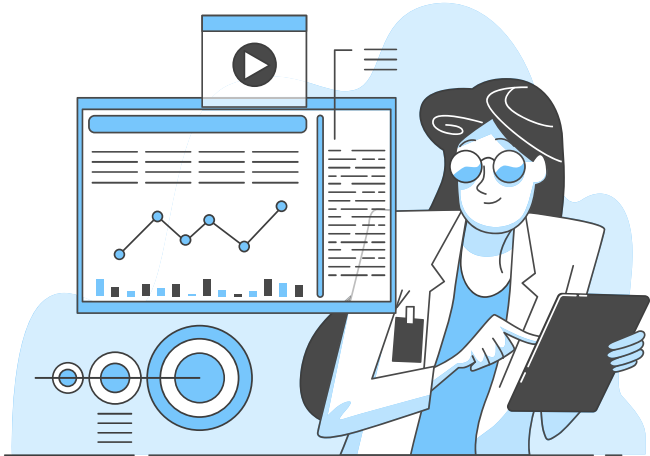
Nội dung

01 Functions (Hàm)

02 Try Catch

03 Làm việc với Object nâng cao

04 Làm việc với Array nâng cao



01

Functions (Hàm)

1.1. Kiến thức tổng quan

- **Hàm** là một **chương trình con** giúp **thực thi** một **công việc cụ thể**.
- Có **2 loại** hàm:
  - Hàm **built-in**: Hàm có sẵn trong Javascript.
    - console.log()
    - alert()
    - prompt()
    - ...
  - Hàm **tự định nghĩa**: Là hàm do **chúng ta tự viết** ra.
    - Ví dụ hàm tính toán cộng, trừ, nhân, chia.

1.1. Kiến thức tổng quan

- Cú pháp hàm tự định nghĩa:

```
function tenHam(thamSo1, thamSo2, ...) {
  // Code
}
```

- Cú pháp gọi hàm:

```
tenHam(doiSo1, doiSo2, ...);
```

1.2. Các loại hàm (các cách viết hàm)

- **Declaration function (Hàm định nghĩa)**
  - **Có** tính **hoisting** (nghĩa là gọi được hàm trước lúc khai báo hàm đó).
  - **Có** đối tượng **arguments**.
  - Cú pháp:

```
function tenHam(thamSo1, thamSo2, ...) {
  // Code
}
```

1.2. Các loại hàm (các cách viết hàm)

- **Expression function (Hàm biểu thức)**
  - **Không** có tính **hoisting** (Nếu gọi hàm trước lúc khai báo hàm thì sẽ gặp lỗi).
  - **Có** đối tượng **arguments**.
  - Cú pháp:

```
var tenHam = function (thamSo1, thamSo2, ...) {
  // Code
}
```

1.2. Các loại hàm (các cách viết hàm)

- **Arrow function (Hàm mũi tên)**
  - **Không** có tính **hoisting** (Nếu gọi hàm trước lúc khai báo hàm thì sẽ gặp lỗi).
  - **Không** có đối tượng **arguments**.
  - Cú pháp:

```
var tenHam = (thamSo1, thamSo2, ...) => {
  // Code
}
```

1.3. Ví dụ về hàm

- Ví dụ 1: Tính tổng các số chẵn của mảng. Mảng như sau:

```
var mang = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
```

- Ví dụ 2: Viết chương trình cho người dùng nhập vào số n lớn hơn 0, sau đó dựa vào n tính giá trị của biểu thức:

```
bieuThuc = 1/n + 2/n + 3/n + ... + n/n
```

02

Try Catch

02. Try Catch

- **Try catch** là một khối lệnh **dùng để bắt lỗi** chương trình.
- Sử dụng try catch khi muốn chương trình không bị dừng khi một lệnh nào đó bị lỗi.
- Cú pháp:

```
try {
  // Chạy vào đây đầu tiên
} catch (error) {
  // Nếu lỗi chạy vào đây
} finally {
  // Luôn luôn chạy vào đây và chạy sau cùng
}
```

03

Làm việc với Object nâng cao

3.1. Cách thêm 1 key mới vào object

- Cú pháp: **tenObject.tenKey = value;**
- Hoặc: **tenObject["tenKey"] = value;**
- Ví dụ:

```
var infoUser = {
  name: "Le Van A"
}
```

```
// Cách 1
console.log(infoUser); // Chưa có key phone
infoUser.phone = "0123456789"; // Thêm key phone cách 1
```

```
// Cách 2
infoUser["phone"] = "0123456789"; // Thêm key phone cách 2
console.log(infoUser); // Có thêm cả key phone
```

3.2. Cách xóa 1 key khỏi object

- Cú pháp: **delete tenObject.tenKey;**
- Ví dụ:

```
var infoUser = {
  name: "Le Van A",
  phone: "0123456789"
}
```

```
delete infoUser.phone;
```

04

Làm việc với Array nâng cao

4.1. forEach()

- Để duyệt qua mỗi phần tử của mảng và thực hiện một hành động nào đó.
- Cú pháp:

```
arr.forEach(function (currentValue, index, array) {
  // code xử lý
});
```

- Trong đó:
  - **currentValue**: phần tử hiện tại đang được xử lý của array.
  - **index**: chỉ số của phần tử hiện tại đang được xử lý của array.
  - **array**: mảng hiện tại đang gọi hàm forEach().

Minh họa

4.2. every()

- Kiểm tra tất cả các phần tử của một mảng phải thỏa mãn một điều kiện gì đó.
- Cú pháp:

```
arr.every(function (currentValue, index, array) {
  // code xử lý
});
```

Minh họa

4.3. some()

- Kiểm tra chỉ cần một phần tử của một mảng thỏa mãn một điều kiện gì đó là được.
- Cú pháp:

```
arr.some(function (currentValue, index, array) {
  // code xử lý
});
```

Minh họa

4.4. find()

- Tìm xem trong mảng có giá trị giống với giá trị đang cần tìm không?
  - Nếu có thì trả về chính phần tử của mảng đó.
  - Nếu không có thì trả về undefined.
- Chỉ tìm được 1 phần tử trong mảng.
- Cú pháp:

```
arr.find(function (currentValue, index, array) {
  // code xử lý
});
```

Minh họa

4.5. filter()

- Giống hàm find().
- Tìm được nhiều phần tử trong mảng.
- Trả về một mảng các phần tử đã tìm được.
- Cú pháp:

```
arr.filter(function (currentValue, index, array) {
  // code xử lý
});
```

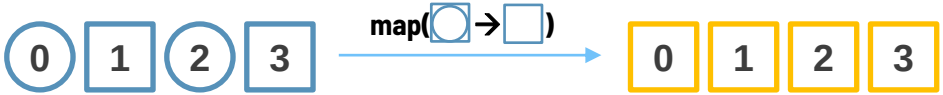
Minh họa

4.6. map()

- Lặp qua từng phần tử của mảng và trả về một mảng mới.
- Số lượng phần tử bằng mảng cũ.
- Giá trị trả về được quyết định bởi từ khóa return.
- Cú pháp:

```
arr.map(function (currentValue, index, array) {
  // code xử lý
});
```

Minh họa



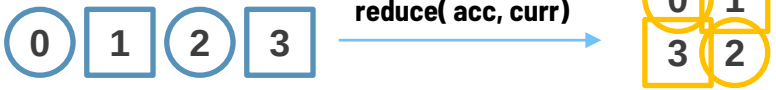
4.7. reduce()

- Hàm reduce() duyệt qua từng phần tử trong mảng để tính toán và trả về một giá trị cuối cùng.
- Cú pháp:

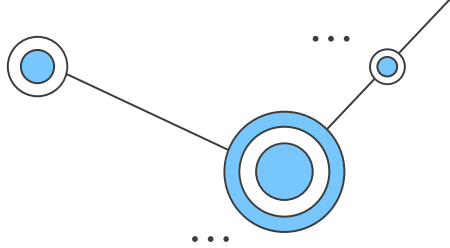
```
arr.reduce(function (accumulator, currentValue, currentIndex, array) {
  // code xử lý
}, initialValue);
```

- Trong đó:
  - **accumulator**: là giá trị của lệnh return cho mỗi lần lặp.
  - **initialValue**: giá trị khởi tạo ban đầu (không bắt buộc).

Minh họa



Bài tập



- **Link bài tập**: https://dacavn.notion.site/B-i-t-p-b-i-07-Javascript-c-b-n-Ti-t-3-b62c38712b4d4bdd8c4269171395f705?pvs=4