Hàm (function)

Một Javasript function là một đoạn code được thiết kế để làm một nhiệm vụ riêng biệt.

Javascript function được thực thi khi gọi nó.

Hàm trong Javascript chia làm 2 loại

1. Khai báo hàm (Function Declaration)
2. Biểu thức hàm (Function Expression)

1. Khai báo hàm (Function Declaration)

Với cách này thì function được **Hoisting**. Và Javascript cho phép chúng ta gọi một hàm trước khi hàm đó được khai báo.

```
hoisted() // Output: "This function has been hoisted."
function hoisted() {
  console.log('This function has been hoisted.')
}
```

2. Biểu thức hàm (Function Expression)

Tuy nhiên với cách khai báo function kiểu này thì sẽ không được hoisting

```
expression() //Output: "TypeError: expression is not a function
var expression = function () {
  console.log('Will this work?')
}
```

Giải thích: Biến `var expression` vẫn được **hoisting** và được đẩy lên trên cùng của scope nhưng chỉ là khai báo mà thôi, nó không được gán cho hàm! Vì thế nó sẽ ném ra lỗi `TypeError`.

```
expression() //Output: "ReferenceError: Cannot access 'expression' before
initialization
let expression = function () {
  console.log('Will this work?')
}
```

Giải thích: khi bạn gọi hàm `expression()` trước khi nó được khai báo. Trong JavaScript, khi sử dụng let hoặc const để khai báo biến, chúng không được "hoisting" (đưa lên đầu) giống như var. Do đó, khi bạn gọi `expression()` trước khi `let expression` được khai báo, JavaScript không biết `expression` là gì và báo lỗi.

Lưu ý khai báo như thế này thì sẽ chạy luôn function và gán `handle` là giá trị mà function return:

```
const handle = (function () {
  console.log('Run here')
})()
```

3. IIFE (Immediately Invokable Function Expression)

IIFE là khởi tạo một function và thực thi ngay lập tức sau đó.

```
;(function () {
  let a = 1
  let b = 2
  console.log('a + b = ' + (a + b))
})()
```

Chúng ta nên có dấu chấm phẩy `;` trước IIFE để tránh trường hợp làm run một function ngoài ý muốn, ví dụ

```
const handle = function () {
  console.log('hello')
}

(function () {
  console.log('IIFE')
})()
```

sẽ giống như chúng ta vừa khai báo handle rồi chạy luôn function với tham số đầu vào là một function IIFE => gây nên lỗi không mong muốn. Vì thế nên thêm dấu `;` trước IIFE.

```
const handle = (function () {
  console.log('hello')
})(function () {
  console.log('IIFE')
})()
```

4. Hàm ẩn danh (Anonymous function)

Hàm ẩn danh là hàm không tên. Nếu bạn để ý thì vế bên phải biểu thức hàm là một **anonymous function**, hay **IIFE** cũng thực thi một hàm ẩn danh. Ngoài ra hàm ẩn danh còn xuất hiện ở **callback**

function bên trong `setTimeout` là một hàm ẩn danh

```
setTimeout(function () {
  console.log('Sau 1s thì sẽ in ra dòng này')
}, 1000)
```

hoặc sử dụng biểu thức hàm:

```
let anonymousFunc = function() {
  console.log('This is an anonymous function.');
};

anonymousFunc(); // Gọi hàm ẩn danh
```

hay sử dụng biểu thức lambda (arrow function):

```
let anonymousFunc = () => {
  console.log('This is an anonymous function.');
};

anonymousFunc(); // Gọi hàm ẩn danh
```

5. Hàm rút gọn - hàm mũi tên (Arrow function)

Hàm rút gọn ngắn hơn biểu thức hàm (function expression) và không phụ thuộc this. Áp dụng tốt cho hàm ẩn danh (anonymous function) nhưng không thể dùng làm hàm khởi tạo

```
const handleClick = () => {
  // thực hiện gì đó
}
```

```
const returnObject = () => ({
  name: "LongDC"
})
console.log(returnObject())
```

Lưu ý với arrow function:

- Không có `this`, sẽ học ở bài tiếp theo
- Không được gọi với `new`
- Cũng không có super, chúng ta sẽ học về nó trong bài kế thừa class

6. Phân biệt parameter (tham số) vs argument (đối số)

```
// a, b là tham số
function sum(a, b) {
  return a + b
```

```
}
// 1,2 là đối số
sum(1, 2)
```

7. Tham số mặc định (default parameter)

```
// 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
```

8. 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); // [...argument]
-> Spread operator
}
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));
```

9. 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
```

10. Constructor function (Buổi class)

```
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');
bob.sayHello(); // My name is Bob
```