

LẬP TRÌNH ECMASCRIPT

HÀM VÀ ĐỐI TƯỢNG

- ⦿ Hiểu và sử dụng hàm Arrow Function
- ⦿ Giải thích con trỏ 'this' trong Arrow Function
- ⦿ Nắm được khái niệm Default Parameters
- ⦿ Nắm được khái niệm Generator Function

- 📖 Giới thiệu Arrow function
- 📖 Làm việc với 'this' trong Arrow function
- 📖 Tổng quan Default Parameters
- 📖 Giới thiệu Generators Functions

PHẦN 1: GIỚI THIỆU ARROW FUNCTION

ARROW FUNCTION LÀ GÌ?

- ❑ **Arrow Function** lần đầu tiên được giới thiệu trong ES6
- ❑ Là cú pháp ngắn gọn hơn [hàm biểu thức](#) (function expression), sử dụng mũi tên =>
- ❑ Cú pháp:
 - (param1, param2, ..., paramN) => { statements }
 - *(param1, param2, ..., paramN) => expression*
 - // tương đương với: *(param1, param2, ..., paramN) => { return expression; }*

ES5

```
[1, 2, 3].map(function(num) {  
    return num * 2;  
});
```

ES6

```
[1, 2, 3].map(num => num * 2);
```

- ❑ Bạn có thể truyền **nhều đối số** trong hàm bằng cách sử dụng dấu ngoặc đơn xung quanh đối số nếu nhiều hơn một đối số

ES5

```
var sum = function (num1, num2) {  
    return num1 + num2;  
};
```

ES6

```
var sum = (num1, num2) => num1 + num2;
```

- ❑ Ngoài ra, nếu hàm **không có đối số**. Bạn bắt buộc phải bao gồm dấu ngoặc đơn (cú pháp giống như đối với nhiều đối số).

```
var doNothing = () => {};
```

- ❑ Nếu có nhiều hơn một câu lệnh bạn cần phải sử dụng dấu ngoặc nhọn {...} để bao bọc phần thân hàm.

ES5

```
[1, 2, 3, 4].map(function (num) {  
    var multiplier = 2 + num;  
    return num * multiplier;  
});
```

ES6

```
[1, 2, 3, 4].map(num => {  
    var multiplier = 2 + num;  
    return num * multiplier;  
});
```

Lưu ý: Khi một hàm có nhiều đối số và nhiều câu lệnh, không có sự khác biệt đáng kể nào với hàm bình thường được khai báo bình thường (*name functions*).

❑ Trường hợp đặc biệt, một hàm rút gọn muốn trả về kết quả là một đối tượng thì phần thân hàm phải bọc bằng dấu ngoặc đơn ()

❑ Ví dụ

ES5

```
var getSomeObject = function (id) {  
    return {  
        key: id,  
        value: "unknown"  
    };  
};
```

ES6

```
var getSomeObject = id => (  
    { key: id, value: "unknown" }  
);
```

Việc bọc object trong dấu ngoặc đơn, thông báo cho hàm biết mã trong dấu ngoặc nhọn là một đối tượng, chứ không phải là thân hàm.

- ❑ Immediately-invoked function expressions (IIFEs) là một hàm Javascript ẩn danh, nó được chạy ngay sau khi nó được định nghĩa.

ES5

```
let person = function(message) {  
  return {  
    getGreeting: function() {  
      return message;  
    }  
  };  
}("Hell World!");
```

ES6

```
let person = ((message) => {  
  return {  
    getGreeting: function() {  
      return message;  
    }  
  });  
})("Hell World!");
```

Để rút ngắn code bằng cách sử dụng các hàm mũi tên, bạn phải đặt hàm mũi tên trong dấu ngoặc đơn.

▶ DEMO

SỬ DỤNG ARROW FUNCTION

- ❑ Đề bài: Cho giao diện như hình, viết lại hàm myConcat() bằng cách sử dụng arrow function gán nội dung của arr2 vào arr1.

```
var myConcat = function(arr1, arr2) {  
    "use strict";  
    return arr1.concat(arr2);  
};
```

```
console.log(myConcat([1, 2], [3, 4, 5]));
```

SỬ DỤNG ARROW FUNCTION

- ❑ Giải pháp: Chúng ta thay thế var bằng const và bỏ từ khóa function.

```
const myConcat = (arr1, arr2) => {  
    "use strict";  
    return arr1.concat(arr2);  
};  
  
console.log(myConcat([1, 2], [3, 4, 5]));
```

PHẦN 2: THIS TRONG ARROW FUNCTION

Ý NGHĨA THIS TRONG ARROW FUNCTION

- ❑ Trong javascript **this** là ngữ cảnh bọc quanh nó, thông thường mỗi đối tượng đều có **this**.
- ❑ Đặc biệt arrow function không có **this** (ngữ cảnh)
- ❑ Cùng xem ví dụ:

```
let person = {  
  name: "adam",  
  funcA: () => { console.log(this.name, this) },  
  funcB: function () { console.log(this.name) }  
}
```

```
person.funcA();  
// kết quả: undefined {}  
person.funcB();  
// kết quả: adam
```

- ❑ Trong ví dụ trên: **this** của method funcA không được tìm thấy nên trả về kết quả undefined

Ý NGHĨA THIS TRONG ARROW FUNCTION

- ❑ Arrow function không tạo ra ngữ cảnh **this** của riêng hàm, do vậy **this** có ý nghĩa trong ngữ cảnh bọc quanh nó

```
function Counter(){
  this.number = 0;

  setInterval(() => {
    this.number++;
    console.log(this.number)
  }, 1000);
}

var p = new Counter();
```

- ❑ Trong ví dụ: arrow function không có **this** nên sẽ lấy **this** của Regular function gần nhất là function Counter

PHẦN 3: DEFAULT PARAMETERS

DEFAULT PARAMETER LÀ GÌ?

- ❑ Default Parameter là tính năng mới của ES6, nó có tác dụng khai báo giá trị mặc định tại tham số

```
function multiply(a, b = 1) {  
  return a * b;  
}
```

```
console.log(multiply(5, 2));  
// kết quả: 10
```

```
console.log(multiply(5));  
// kết quả: 5
```

Trong ví dụ trên biến b được gán giá trị mặc định bằng 1.
Nếu chỉ truyền 1 tham số thì hàm sẽ lấy giá trị mặc định của b

- ❑ Giá trị mặc định của tham số bằng undefined
- ❑ Ví dụ:

```
function sum(a, b) {  
  return a * b;  
}  
  
sum(5)  
// 5 * undefined => kết quả: NaN
```

Trong ví dụ trên khi chỉ truyền 1 đối số (5) lúc này giá trị của b là undefined
Hàm sẽ trả về kết quả NaN (Not a Number)

- ❑ Giải pháp: kiểm tra giá trị của số b trước khi hàm trả về

```
function sum(a, b) {  
  b = typeof b !== 'undefined' ? b : 1  
  return a * b;  
}  
  
sum(5)  
// 5 * 1 => kết quả: 5
```

Trong ví dụ trên khi chỉ truyền 1 đối số (5) lúc này giá trị của b là undefined
Hàm sẽ trả về kết quả NaN (Not a Number)

▶ DEMO

SỬ DỤNG DEFAULT PARAMETER

- ❑ Đề bài: Thay đổi hàm increment() bằng cách sử dụng tính năng “default parameter” để thêm 1 vào **number** nếu đối số value không được truyền vào.

```
const increment = (number, value) => number + value;
```

```
increment(5, 2); // Kết quả: 7
```

```
increment(5); // Kết quả mong muốn : 6
```

SỬ DỤNG DEFAULT PARAMETER

❑ Lời giải: truyền giá trị `value = 1` vào tham số

```
const increment = (number, value = 1) => number + value;
```

```
increment(5, 2); // Kết quả: 7
```

```
increment(5); // Kết quả : 6
```

PHẦN 3: TỔNG QUAN GENERATOR FUNCTION

GENERATOR FUNCTION LÀ GÌ?

- ❖ Generator function (`function*`) là một hàm có thể thoát và sau đó gọi lại lần nữa. Giá trị của biến trong các lần gọi được lưu lại trong các lần gọi tiếp theo.
- ❖ Trả về một đối tượng [Generator](#)
- ❑ Cú pháp

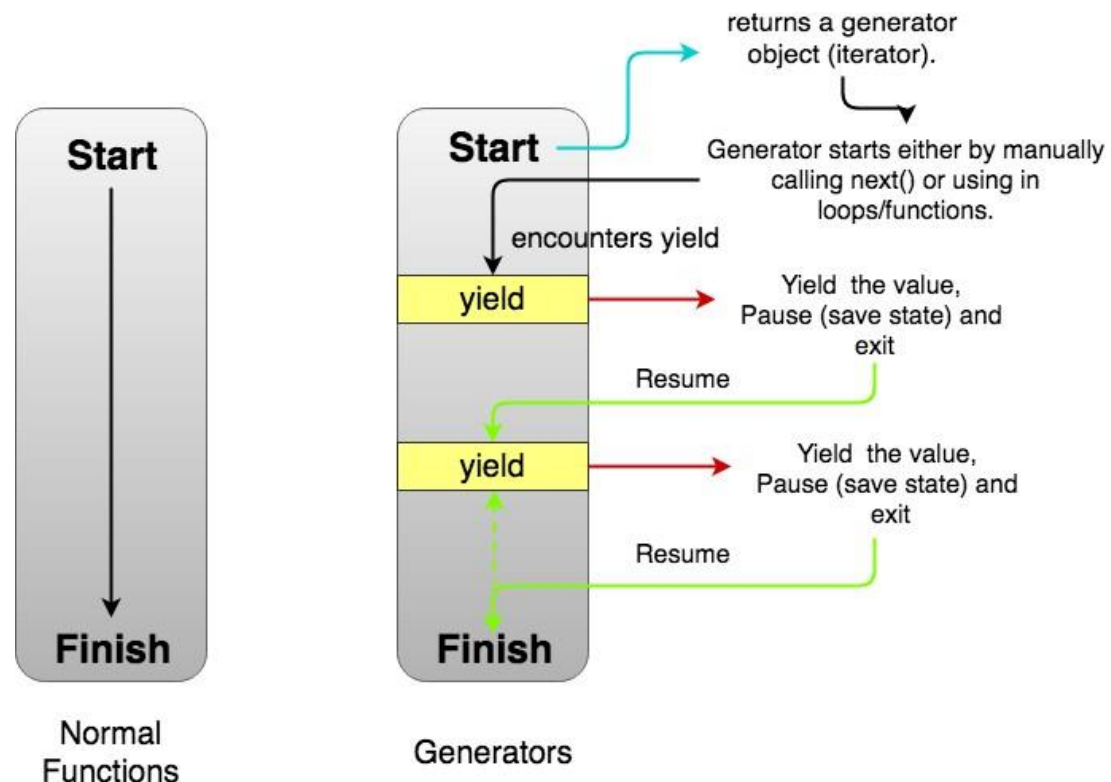
```
function* name([param[, param[, ... param]]) { statements }
```

Name: Tên hàm

Param: Các tham số truyền vào cho hàm.

Statements: Các câu lệnh bên trong hàm

- ❖ Khi chúng ta gọi một function* nó không trả về các kiểu dữ liệu cơ bản mà đẩy về một iterator object .
- ❖ Hàm next() được sử dụng để truy xuất dữ liệu sau mỗi bước chạy
- ❖ Lúc này các lệnh bên trong hàm được thực thi cho đến khi gặp từ khóa yield
- ❖ Sau câu lệnh yield là giá trị sẽ trả về
- ❖ Khi một generator kết thúc, các câu gọi next tiếp theo sau sẽ không được thực thi, nó chỉ trả về object có dạng: {value: undefined, done: true}.



- ❖ Giải thích: Các lệnh trong hàm `swimming()` được thực thi cho đến khi gặp câu lệnh `yield`.
- ❖ Khi instance **S** gọi phương thức `next()` hàm generator sẽ chạy và trả về object iterator gồm property là `value` và `done`. `done` có kiểu Boolean xác định `yielded` trả về đã là cuối cùng chưa.

```
function* swimming (number) {  
    yield number;  
    yield number + 2  
    yield number + 5  
}
```

```
let s = swimming(10)  
  
console.log(s.next())  
// kết quả: { value: 10, done: false }  
console.log(s.next())  
// kết quả: { value: 12, done: false }  
console.log(s.next())  
// kết quả: { value: 17, done: false }
```

▶ DEMO

SỬ DỤNG GENERATOR FUNCTION

- ❑ Đề bài: Tạo một vòng lặp đếm ngược từ 9 đến 1. Sử dụng generator function!

```
let getCountdownIterator = // Code của bạn sẽ ở đây
```

```
console.log( [...getCountdownIterator()] );  
// Kết quả: [9, 8, 7, 6, 5, 4, 3, 2, 1]
```

SỬ DỤNG GENERATOR FUNCTION

□ Giải pháp:

- ❖ Generator function sử dụng yields để chạy từng bước từ 9 đến 1. Cú pháp (...) spread operator sẽ bóc tách tất cả giá trị.
- ❖ Sau đó Generator Function sẽ trả về các giá trị đến vị trí cuối cùng của vòng lặp

```
let getCountdownIterator = function *() {  
    let i = 10;  
    while( i > 1 ) {  
        yield --i;  
    }  
}
```

```
console.log( [ ...getCountdownIterator() ] );  
// Kết quả : [9, 8, 7, 6, 5, 4, 3, 2, 1]
```

- ☑ Hiểu và sử dụng hàm Arrow Function
- ☑ Giải thích con trỏ 'this' trong Arrow Function
- ☑ Nắm được khái niệm Default Parameters
- ☑ Nắm được khái niệm Generator Function

A rustic, handmade-style image featuring a rectangular piece of light brown cardboard or kraft paper. The words "Thank you!" are written on the tag in a black, cursive script. A black string is tied through a hole on the left side of the tag. Three white daisies with bright yellow centers are scattered around the tag: one is in sharp focus to the right of the tag, while two others are in the background, slightly out of focus. The entire scene is set against a background of light-colored, weathered wood with visible grain and texture.

Thank
you!