Bài 2

Tổng Quan ES6

Module: Web app building with JS

Mục tiêu

- Trình bày khái niệm tổng quan về ES6
- Trình bày cách hoạt động của let & const
- Trình bày được Arrow Functions
- Triển khai được Default Parameters
- Triển khai được vòng lặp for
- Trình bày được Spread Attributes
- Trình bày được cơ chế module
- Sử dụng lambda expression
- Sử dụng được Maps, Sets
- Sử dụng được Static methods
- Sử dụng được Getter và Setter

Tổng quan ES6

- ECMAScript 2015 hay ES2015 là một bản cập nhật quan trọng cho ngôn ngữ lập trình JavaScript
- ES2015 thường được gọi là ES6
- Một số tính năng mới như: các biến phạm vi, vòng lặp mới...
- Các thư viện framework hiện đại của JavaScript như React.js, Angular, Vue rất hay sử dụng các tính năng mới này.

Let & const

- `let` là phạm vi khối `{ }` (block scope)
- `var` có phạm vi trong hàm (function scope) và hành vi hoisting trong JavaScript xảy ra

```
// Cú pháp ES5
for (var i = 0; i < 5; i++) {
    console.log(i); // 0,1,2,3,4
}
console.log(i); // 5

// Cú pháp ES6
for (let i = 0; i < 5; i++) {
    console.log(i); // 0,1,2,3,4
}
console.log(i); // undefined
```

- Từ khóa `const` mới được giới thiệu trong ES6 **định nghĩa các hằng số**
- **Hằng số** ở chế độ **chỉ đọc**

Sự khác biệt giữa var, let và const?

Var

với từ khóa var chúng ta có thể khai báo đa dạng các kiểu biến như number, string, boolean, etc. Trừ trường hợp được khai báo bên trong 1 function (khi đó biến var sẽ có scope là function/locally scoped), biến var sẽ có scope là globally scoped. Đặc biệt, biến var còn có thêm tính chất **hoisting**: nghĩa là dù khai báo ở đâu thì biến đều sẽ được đem lên đầu scope trước khi code được thực hiện.

Lấy ví dụ:

```
console.log (greeting);
var greeting = "say hello";
```

sẽ được biên dịch là:

```
var greeting;
console.log(greeting); // greeting is undefined
greeting = "say hello";
```

Sự khác biệt giữa var, let và const?

Let

biến let được khai báo sẽ có scope là block scoped chứ không phải globally hay locally scoped, ví dụ:

```
let greeting = "say Hi";
let times = 4;

if (times > 3) {
    let hello = "say Hello instead";
    console.log(hello); // "say Hello instead"
 }
console.log(hello); // hello is not defined
```

Sự khác biệt giữa var, let và const?

Const

Trong biến const nếu trường hợp kiểu của biến là **primitive (bao gồm string, number, boolean, null, và undefined)** thì chúng ta sẽ không thể tái khai báo hay cập nhật giá trị mới để thay thế cho giá trị trước đó của biến.

```
const greeting = "say Hi";
greeting = "say Hello instead"; // error : Assignment to constant variable.
const greeting = "say Hi";
const greeting = "say Hello instead"; // error : Identifier 'greeting' has already been declared
```

Đối với trường hợp kiểu biến **là reference (bao gồm object, array, và function)** thì tuy không thể tái khai báo hay cập nhật giá trị của biến nhưng chúng ta vẫn có thể cập nhật giá trị cho thuộc tính của biến đó.

```
const greeting = {
    message : "Hello",
    number : "five"
}
greeting.message = "say Hello instead";
console.log(greeting); // {message:"say Hello instead",number:"five"}
```

ARROW FUNCTION (1)

- Cung cấp cú pháp ngắn gọn hơn để viết **biểu thức hàm**
- Các Arrow Function được định nghĩa bằng cú pháp mới, ký hiệu suy ra `=>`

```
// Cách 1: sử dụng Anonymous Function
var sum = function(a) {
    return a + 100;
}
console.log(sum(2, 3)); // 5

// Cách 2: Sử dụng Arrow function
var sum = (a) => {
  return a + 100;
}
```

ARROW FUNCTION (2)

- 1 tham số, nếu biểu thức tính toán là đơn giản thì không cần câu lệnh return.

```
param => expression
```

- Nhiều tham số, nếu biểu thức tính toán là đơn giản thì không cần câu lệnh return.

```
(param1, paramN) => expression
```

- Nhiều câu lệnh trong thân hàm, trong trường hợp này cần sử dụng dấu {} và câu lệnh return:

```
param => {
  let a = 1;
  return a + param;
}
```

- Nhiều tham số cần dùng dấu (). Nhiều câu lệnh cần dùng ngoặc {} và câu lệnh return:

```
(param1, paramN) => {
   let a = 1;
   return a + param1 + paramN;
}
```

Giá trị mặc định

- Trong ES6, bạn có thể chỉ định các **giá trị mặc định** cho các tham số
- Nếu không có đối số nào được cung cấp cho hàm khi nó được gọi thì các giá trị tham số mặc định này sẽ được sử dụng

```
function greet(name, greeting, message = greeting + ' ' + name) {
  return [name, greeting, message]
}

greet('David', 'Hi')                    // ["David", "Hi", "Hi David"]
greet('David', 'Hi', 'Happy Birthday!')  // ["David", "Hi", "Happy Birthday!"]
```

Vòng lặp for

- ES6 giới thiệu vòng lặp **`for of`** sử dụng để duyệt qua từng phần tử trong mảng hoặc đối tượng.
- Các vòng lặp có thể sử dụng gồm: for of, for in, foreach, for

```
const iterable = [10, 20, 30];

for (const value of iterable) {
  console.log(value);
}

// 10
// 20
// 30
```

Vòng lặp for

Thực hành Vòng lặp for

Spread Attributes

- Giúp bạn làm việc với các cấu trúc dữ liệu nhanh và gọn hơn.
- Cú pháp spread đơn giản được biểu diễn bởi dấu 3 chấm: ...
- Spread Syntax cho phép duyệt qua các phần tử và truyền vào phương thức như các đối số.

Ví dụ:

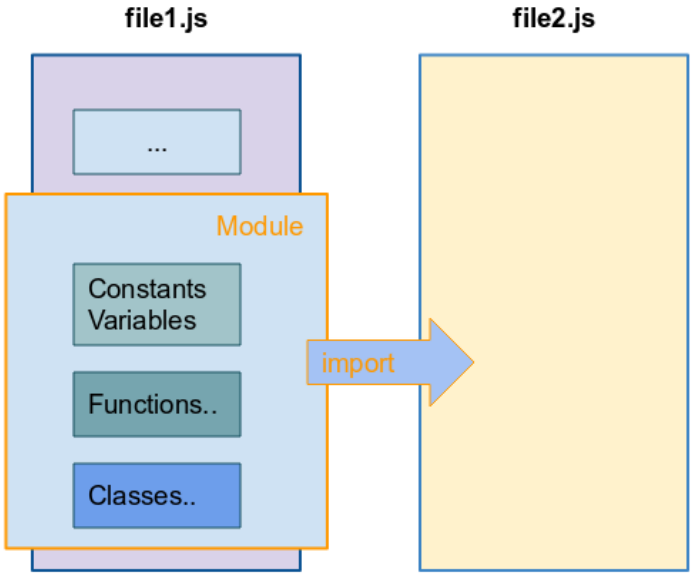
```
// Create an Array
const tools = ['hammer', 'screwdriver']

const otherTools = ['wrench', 'saw']

// Unpack the tools Array into the allTools Array
const allTools = [...tools, ...otherTools]

console.log(allTools)
```

Spread Attributes

Thực hành Spread Attr

Module

- ECMAScript 6 giới thiệu ES6 **`Module Syntax`** giúp các lập trình viên module hóa (modularize) code.
- Có thể viết code trên các tập tin riêng rẽ.
- Trên tập tin này có thể xuất (export) dữ liệu cần thiết dưới dạng các module, các tập tin khác có thể nhập (import) các module của tập tin đó để sử dụng.



Module

Thực hành Module

Lambda Expression

forEach() dùng để duyệt mỗi phần tử trong mảng, **không duyệt mảng rỗng**

Cú pháp:

array.forEach(*function(currentValue, index, arr), thisValue*)

Ví dụ:

```
let text = "";
const fruits = ["apple", "orange", "cherry"];
fruits.forEach(myFunction);
document.getElementById("demo").innerHTML = text;
function myFunction(item, index) {
  text += index + ": " + item + "<br>";
}
```

Lambda Expression

```
let kids = [
    {id: 1, name: "a", age: 3},
    {id: 2, name: "b", age: 2},
    {id: 3, name: "c", age: 7},
    {id: 4, name: "d", age: 4},
    {id: 5, name: "e", age: 8}
    ];
```

Chúng ta tìm tất cả các bé còn đang học mẫu giáo và đưa vào danh sách.

```
const mau_giao = [];
    kids.forEach(kid => {
        if (kid.age < 6) {
            mau_giao.push({
                id: kid.id,
                name: kid.name
            })
        }
    });
console.log(mau_giao);
```

Lambda Expression

map() tạo một array mới bằng cách **gọi một hàm** cho tất cả **từng phần tử**, gọi **mỗi hàm một lần** cho **mỗi phần tử**, **không** áp dụng với các phần tử rỗng và **không làm thay đổi mảng cũ**

Ví dụ:

```
const numbers = [4, 9, 16, 25];
const newArr = numbers.map(Math.sqrt)

// newArr = [2, 3, 4, 5]
```

Lambda Expression

```
const numbers = [65, 44, 12, 4];
const newArr = numbers.map(myFunction)

function myFunction(num) {
  return num * 10;
}

// newArr = [650, 440, 120, 40]
```

Lambda Expression

```
const persons = [
  {firstname : "Malcom", lastname: "Reynolds"},
  {firstname : "Kaylee", lastname: "Frye"},
  {firstname : "Jayne", lastname: "Cobb"}
];

persons.map(getFullName);
function getFullName(item) {
  return [item.firstname,item.lastname].join(" ");
}

// Malcom Reynolds,Kaylee Frye,Jayne Cobb
```

Lambda Expression

filter() tạo một array mới với các phần tử thỏa mãn một số điều kiện nhất định, không thực thi với các phần tử rỗng, không làm thay đổi giá trị mảng ban đầu

Ví dụ:

```
const ages = [32, 33, 16, 40];
const result = ages.filter(checkAdult);
function checkAdult(age) {
  return age >= 18;
}
// result = [32, 33, 40]
```

Lambda Expression

```
let kids = [
    {id: 1, name: "a", age: 3},
    {id: 2, name: "b", age: 2},
    {id: 3, name: "c", age: 7},
    {id: 4, name: "d", age: 4},
    {id: 5, name: "e", age: 8}
    ];
```

```
const kids2 = kids
 .filter(kid => kid.age < 11 && kid.age > 5)
 .map(kid => ({
    id: kid.id,
    name: kid.name
}));
 console.log(kids2);
```

ES6 – Lambda Expression

Bài tập: In ra tất cả mã khoa học của các loài chim trong mảng dữ liệu.

```
const birds = [
        {"ID": "DV8", "Name": "Eurasian Collared-Dove", "Type": "Dove" },
        {"ID": "HK12", "Name": "Bald Eagle", "Type": "Hawk" },
        {"ID": "HK6", "Name": "Cooper's Hawk", "Type": "Hawk" },
        {"ID": "SP11", "Name": "Bell's Sparrow", "Type": "Sparrow" },
        {"ID": "DV2", "Name": "Mourning Dove", "Type": "Dove" }
];
```

ES6 – SET

Set là một tập hợp lưu trữ các giá trị duy nhất của các kiểu dữ liệu, bạn có thể duyệt qua các phần tử của tập hợp này theo thứ tự như khi thêm vào.

Khởi tạo:

```
const mySet = new Set();
```

ES6 – SET

Ví dụ 1: Duyệt các phần tử trong Set

ES6 – SET

Ví dụ 2: Remove duplicate elements

```
const numbers = [2,3,4,4,2,3,3,4,4,5,5,6,6,7,5,32,3,4,5]
console.log([...new Set(numbers)])
// [2, 3, 4, 5, 6, 7, 32]
```

```
let text = 'India'
const mySet = new Set(text)  // Set(5) {'I', 'n', 'd', 'i', 'a'}
mySet.size  // 5
//case sensitive & duplicate omission
new Set("Firefox")  // Set(7) { "F", "i", "r", "e", "f", "o", "x" }
new Set("firefox")  // Set(6) { "f", "i", "r", "e", "o", "x" }
```

ES6 – SET

Bài tập: Peter cần quản lý danh sách động vật trong sở thú, đôi khi cần thêm 1 loài vật mới vào danh sách, đôi khi cần kiểm tra xem loài vật mới đã có trong danh sách chưa, có lúc thì cần xóa một loài động vật khỏi danh sách quản lý của sở thú. Ngoài ra khi có đoàn kiểm tra tới, Peter cần in ra danh sách các loài động vật hiện đang có trong sở thú.

ES6 – Maps

Maps chứa các cặp key-value với key có thể là bất cứ kiểu dữ liệu nào, đồng thời, nó ghi nhớ trình tự thêm vào của các key

Ví dụ:

```
// Create a Map
const fruits = new Map([
  ["apples", 500],
  ["bananas", 300],
  ["oranges", 200]
]);

document.getElementById("demo").innerHTML = fruits.get("apples"); // 500
```

ES6 – Maps

Method	Description
new Map()	Creates a new Map object
set()	Sets the value for a key in a Map
get()	Gets the value for a key in a Map
clear()	Removes all the elements from a Map
delete()	Removes a Map element specified by a key
has()	Returns true if a key exists in a Map
forEach()	Invokes a callback for each key/value pair in a Map
entries()	Returns an iterator object with the [key, value] pairs in a Map
keys()	Returns an iterator object with the keys in a Map
values()	Returns an iterator object of the values in a Map

Property	Description
size	Returns the number of Map elements

ES6 – Maps

Ví dụ: Gộp 2 array với Maps

ES6 – Static

Từ khóa **static** dùng để định nghĩa một phương thức hoặc một thuộc tính trong Lớp.

- Các phương thức tĩnh thường là các hàm tiện ích, chẳng hạn như các hàm để tạo hoặc sao chép các đối tượng
- Các thuộc tính tĩnh hữu ích cho bộ nhớ đệm, cấu hình cố định hoặc bất kỳ dữ liệu nào khác mà bạn không cần sao chép qua các phiên bản.

ES6 – Static

Không thể gọi một phương thức static trên một đối tượng, mà phải dung ở trên lớp của đối tượng đó.

ES6 – Static

Nếu bạn muốn sử dụng đối tượng bên trong phương thức static, bạn có thể gửi nó dưới dạng tham số:

```
class Car {
  constructor(name) {
    this.name = name;
  }
  static hello(x) {
    return "Hello " + x.name;
  }
}
let myCar = new Car("Ford");
document.getElementById("demo").innerHTML = Car.hello(myCar);
```

ES6 – Getter and Setter

Các **Getter** và **Setter** cho phép chúng ta truy cập hoặc thay đổi một thuộc tính của Đối tượng

Ví dụ:

```
class Example {
  get hello() {
    return 'world';
  }
}
const obj = new Example();
console.log(obj.hello);  // world
```

Tóm tắt bài học

- ECMAScript 6 (ES6) cung cấp **1 số tính năng mới** cho JavaScript
- Giúp code trở nên **tường minh và dễ viết** hơn.
- Các thư viện / framework hiện đại của JavaScript như React.js, Angular, Vue rất hay sử dụng các tính năng mới này.