**BÁO CÁO 1: NHỮNG THAY ĐỔI, CẢI TIẾN TỪ ES6 – ES11**

1. **ES6**
2. **Arrow function**

* Trước đây ta thường khai báo hàm dạng như sau:

return function func() {

    //code

}

* Còn bây giờ ta sử dụng => để chỉ ra function :

return () => {

    //code

}

* Việc sử dụng arrow function giúp tiết kiệm thời gian code hơn

1. **Class**

* ES6 cung cấp các lớp cho người dùng sử dụng:

class Student {

    constructor(id, name) {

        this.id = id;

        this.name = name;

    }

    func() {

        //code

    }

}

var objStudent = new Student(1, "Uy");

objStudent.func();

1. **Let and const**

* Tầm vực của let là block scope chứ không phải globally hay locally scoped. Chính vì thế biến let chỉ có hiệu lực trong block code. Ngoài ra biến let không thể tái khai báo
* Const là biến không thể tái khai báo hoặc cập nhật giá trị của biến. Tuy nhiên các thuộc tính của biến vẫn có thể cập nhật được giá trị

1. **Template String**

* Trước đây, khi muốn tạo các string từ các biến ta sử dụng dấu +. Ví dụ

var a = 5;

var str = "a is " + a;

* Bây giờ việc tạo string từ các biến chỉ cần đặt vào trong ${}

var t = 10;

var temp = `t: ${t} `;

1. **Promise**

* Promise là một cơ chế trong JavaScript giúp thực thi các tác vụ bất đồng bộ mà không rơi vào callback hell hay pyramid of doom, là tình trạng các hàm callback lồng vào nhau ở quá nhiều tầng. Ví dụ:

function readFilePromise(path) {

    return new Promise(function(resolve, reject) {

        fs.readFile(path, function(err, data) {

            if(err) {

                reject(err);

            } else {

                resolve(data);

            }

        });

    });

}

readFilePromise('file.txt')

    .then(function(file) {

        console.log(file);

    })

    .catch(function(err) {

        console.log(err);

    })

* Khi .then(…) có return, thì giá trị trả về sẽ được bọc bên trong 1 promise. Cho phép kết nối nhiều promise với nhau. Ví dụ:

promise()

  .then(() => {

    return 'foo'

  })

  .then(result1 => {

    console.log(result1) // 'foo'

    return anotherPromise()

  })

  .then(result2 => console.log(result2)) // `result2` sẽ là kết quả của anotherPromise()

  .catch(err => {})

1. **ES7**
2. **Array.prototypes.include**

* Được sử dụng để kiểm tra phẩn tử có nằm trong array hay không

var arr = [1, 2, 3]

console.log(arr.includes(1)); //true

console.log(arr.includes(0)); //false

1. **Exponentiation operator**

* Là toán tử mũ:

var a = Math.pow(3, 2)

if(3\*\*2 === a) {

    console.log(true);

}

else {

    console.log(false);

}

1. **ES8**
2. **Object.values() và Object.entries()**

***Object.values() :***

* Ta có object sau :

var obj = {

    firstName: "Uy",

    lastName: "Tran"

}

* Với hàm Object.entries(), nó sẽ trả về mảng các cặp key – value của object:
* Tương tự Object.entries(), Object.values() trả về mảng các value của object

1. **String.prototype.padStart() và String.prototype.padEnd()**

* Hai hàm này được sử dụng để thêm chuỗi đệm vào đầu hoặc vào cuối của một string

1. **Async, await**

* Async: Khai báo hàm bất đồng bộ
  + Tự động biến đổi một hàm thông thường thành một Promise
  + Khi gọi tới hàm async nó sẽ xử lý mọi thứ và trả về kết quả trong hàm của nó
  + Async cho phép sử dụng await
* Await: tạm dừng tất cả các hàm async
  + Khi đặt trước Promise, nó sẽ đợi Promise kết thúc và trả về kết quả
  + Await chỉ làm việc với Promise, nó không hoạt động với callback
  + Await chỉ có thể được sử dụng bên trong các function async

1. **ES9**
2. **Async iteration**

* Cú pháp:

for await (let item of listItem) {

    //code

}

1. **Promise.prototype.finally**

* Đây là bước cuối cùng thực hiện trong Promise dù nó được resolve hay reject. Hàm này trả về một Promise. Ví dụ :

let isLoading = true;

fetch(myRequest).then(function(response) {

    var contentType = response.headers.get("content-type");

    if(contentType && contentType.includes("application/json")) {

      return response.json();

    }

    throw new TypeError("Oops, we haven't got JSON!");

  })

  .then(function(json) { /\* process your JSON further \*/ })

  .catch(function(error) { console.log(error); })

  .finally(function() { isLoading = false; });

1. **Rest operator**

* Toán tử “…” được sử dụng để chọn tất cả các phần tử còn lại của mảng. Ví dụ:

const fruits = { orange: 1, melon: 2, apple: 10, banana: 4 }

const { orange, ...r } = fruits;

console.log(r); // {melon: 2, apple: 10, banana: 4 };

1. **ES10**
2. **Optional catch binding**

* Sử dụng try catch mà không có tham số truyền vào. Ví dụ:

try {

    //code

} catch {

    // code

}

1. **Object.fromEntries()**

* Ngược lại với Object.entries(). Object.fromEntries() trả về một Object từ mảng các cặp key – value

const entries = [

    ['foo', 'bar'],

    ['baz', 42]

];

const obj = Object.fromEntries(entries);

console.log(obj);

// expected output: Object { foo: "bar", baz: 42 }

1. **Array.flat()**

* Hàm sử dụng để flat các mảng lồng nhau. Ví dụ:

var arr = [1, 2, [3, 4, [5, 6]]];

arr.flat();

// [1, 2, 3, 4, [5, 6]]

* Nếu truyền vô tham số cho hàm này, nó sẽ là số mảng lồng nhau mà mình muốn flat. Ví dụ :

var arr1 = [1, 2, [[3, 4, [5, 6]]]];

arr1.flat(2);

// [1, 2, [3, 4, 5, 6]]

var arr2 = [1, 2, [3, 4, [5, 6, [7, 8, [9, 10]]]]];

arr2.flat(Infinity);

// [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

1. **Array.flatMap()**

* Đây là hàm flat mảng sau khi map.

let arr = ["it's Sunny in", "", "California"];

arr.map(x => x.split(" "));

// [["it's","Sunny","in"],[""],["California"]]

arr.flatMap(x => x.split(" "));

// ["it's","Sunny","in", "", "California"]

1. **String.trimStart() và String.trimEnd()**

* Xóa các khoảng trắng trước hoặc sau chuỗi. Ví dụ:

var greeting = '   Hello world!   ';

console.log(greeting.trimStart());

// expected output: "Hello world!   ";

console.log(greeting.trimEnd());

// expected output: "   Hello world!";

1. **Dynamic import**

* Ta có static import như sau :

export function getMessage() {

    return "Hello world";

}

import { getMessage } from './mymodule.js';

let message = getMessage();  // output  "Hello world";

* Với dynamic import, nó sẽ trả về một Promise của module request :

import(moduleSpecifier)

1. **globalThis object**
2. **ES11**
3. **Optional chaining ?.**

* Cho phép đọc giá trị nằm sâu bên trong object mà không cần kiểm tra xem có valid hay không
* Ví dụ:

const adventurer = {

    name: 'Alice',

    cat: {

        name: 'Dinah'

    }

};

const dogName = adventurer.dog?.name;

console.log(dogName);

// expected output: undefined

1. **Private field #**

* Bây giờ ta có thể khai báo các thuộc tính private trong class bằng cách thêm dấu #. Ví dụ:

class Foo {

    #b = 15;

    a = 10;

    get() {

      return this.#b;

    }

}

const obj = new Foo();

obj['#b']; // undefined

1. **Undefined JS, Nullish Coalescing ??**

* Với toán tử ?? ta có đoạn code sau :

a() ?? b()

* Nếu a() là null hoặc undefined, toán tử sẽ trả về vế phải. Để đề phòng vế trái là falsy ta có thể sử dụng a() || b()

1. **Top level await**