TERRALOGIC

ES6/ES7

TRƯƠNG QUỐC BẢO

NGUYỄN PHAN TUẤN ĐẠT

What is ES6? Why?

-JavaScript ES6 (also known as ECMAScript 2015 or ECMAScript 6) is the newer version of JavaScript that was introduced in 2015.

-ES6 brings:

+New syntax and new awesome features.

+It allows you to write less code and do more

Global Objects (Array, Date, JSON, Object,..)

[Arrays]

JS

JSON

JavaScript Object Notation

JavaScript Objects

```
let person = {
   firstName: 'John',
   lastName: 'Doe'
};
```

JS

Array

```javascript
var array = [1, 2, 3];
let array = [1, 2, 3];
const array = [1, 2, 3];

let arr = new Array();
arr.push(4, 9, 16)
console.log(arr);

let arr = new Array(4, 9, 16);
console.log(arr[0]);
console.log(arr[1]);
console.log(arr[2]);
```

Date

```javascript
// How to get the date of yesterday, today, and tomorrow
// in JavaScript

const today     = new Date();
const yesterday = new Date(today);
const tomorrow  = new Date(today);

yesterday.setDate(yesterday.getDate() - 1);
tomorrow.setDate(tomorrow.getDate() + 1);

// You can use the DateTimeFormat object to also format it.
const config = {
  year:  'numeric',
  month: 'short',
  day:   '2-digit'
};

const DTF = new Intl.DateTimeFormat('default', config);

console.log(DTF.format(today));

// Returns the following formatted date

← "Apr 09, 2020"
```

Made with ❤

Object

scratch_2.js

```javascript
// #1
var pizza = {
  name: 'Margherita',
  size: 'medium',
  isVegetarian: true
}

// #2
var pizza = new Object();
pizza.name = 'Margherita';
pizza.size = 'meidum';
pizza.isVegetarian = true;

// #3
var anotherPizza = Object.create(pizza);

// #4
function Pizza() {
```

JSON

```
[
  {
    "date":"2013-11-05",
    "locations":{
      "United States":4,
      "Germany":8
    }
  },
  {
    "date":"2013-11-11",
    "locations":{
      "South Africa":9
    }
  },
  {
    "date":"2013-11-12",
    "locations":{
      "Japan":6
    }
  }
]
```

Console Elements Sources Network Performance Memory Application

top Filter All levels

```
> // parse JSON to Javascript object
var text = JSON.parse( '{ "firstName" : "Ramesh", "lastName" : "Fadatare",  "emailId" :
"ramesh@gmail.com",  "age" : "29"  }');
console.log(text);

// convert Javascript object to JSON string
var user = {
    firstName : 'Ramesh',
    lastName : 'Fadatare',
    emailId : 'ramesh@gmail.com',
    age : 29
}
JSON.stringify(user)
console.log(JSON.stringify(user));
```

output

```
▶{firstName: "Ramesh", lastName: "Fadatare", emailId: "ramesh@gmail.com", age: "29"}
{"firstName":"Ramesh","lastName":"Fadatare","emailId":"ramesh@gmail.com","age":29}
```

Compare the differences?

Arrow function

In the ES6 version, you can use arrow functions to create function expressions

This function :

```
// function expression
let x = function(x, y) {
   return x * y;
}
```

Arrow function:

```
// function expression using arrow function
let x = (x, y) => x * y;
```

Class

JavaScript class is used to create an object. Class is similar to a constructor function.

```
class Person {
  constructor(name) {
    this.name = name;
  }
}
```

Keyword class is used to create a class. The properties are assigned in a constructor function.

Now you can create an object

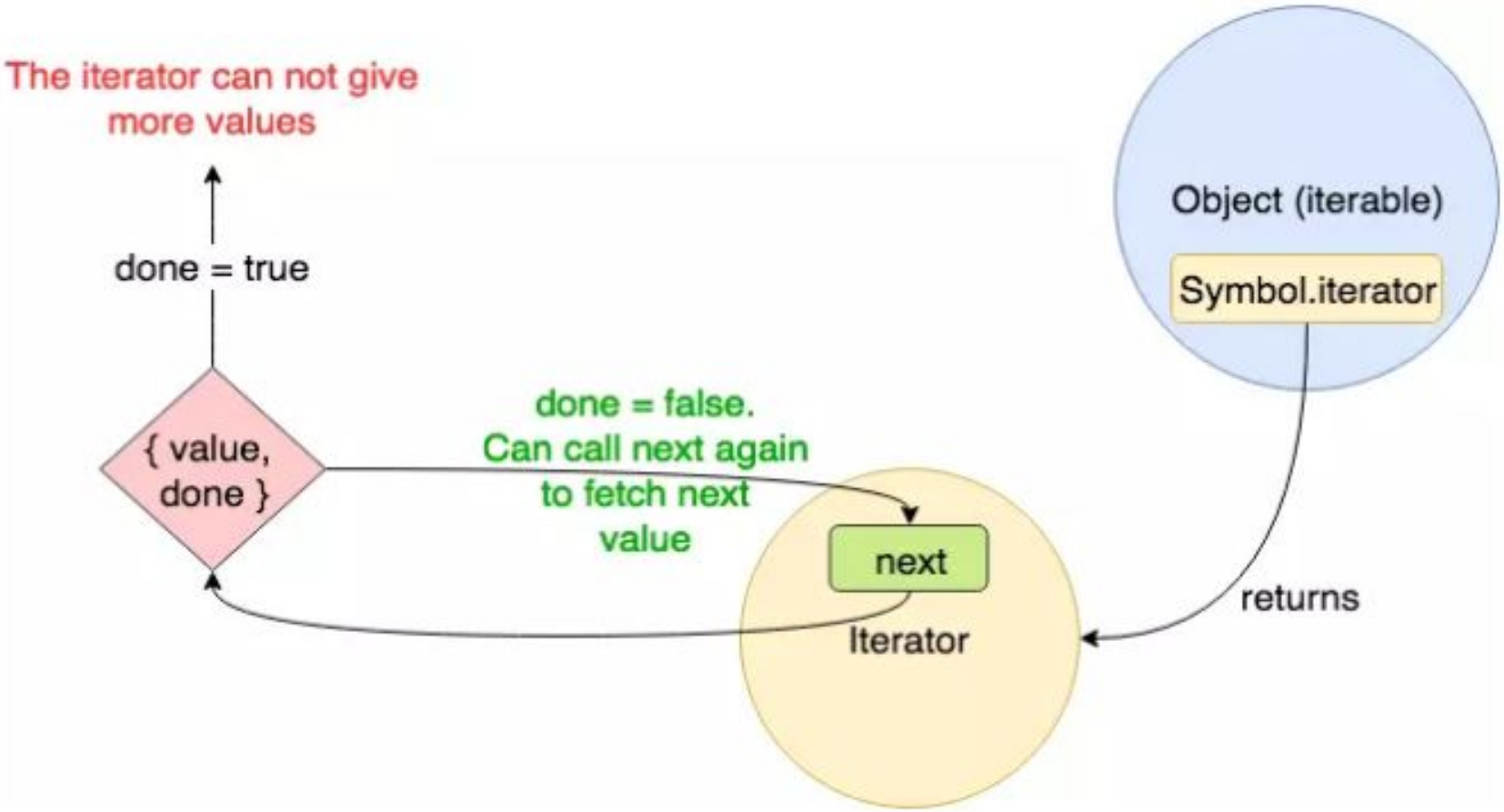
```
class Person {
  constructor(name) {
    this.name = name;
  }
}

const person1 = new Person('John');

console.log(person1.name); // John
```

Iterator

-In JavaScript an iterator is an object which defines a sequence and potentially a return value upon its termination.

-Specifically, an iterator is any object which implements the Iterator protocol by having a next() method that returns an object with two properties

```
1 const numbers = [1, 2, 3, 4, 5];
2 const [first, ,third, ,last] = numbers;
3
4 // is equivalent to
5
6 const numbers = [1, 2, 3, 4, 5];
7 const iterator = numbers[Symbol.iterator]();
8 const first = iterator.next().value
9 iterator.next().value
10 const third = iterator.next().value
11 iterator.next().value
12 const last = iterator.next().value
```


The iterator can not give more values
done = true
{ value, done }
done = false. Can call next again to fetch next value
next
Iterator
returns
Object (iterable)
Symbol.iterator

Shorthand

- Declaring variables
- Assigning values to multiple variables
- Assigning default value
- The ternary operator
- Template Literals
- …..

Declaring variables

```
//Long version
let a;
let b = 1;

//Shorthand
let a, b = 1;
```

Assigning values to multiple variables

```
//Long version
x = 1;
y = 2;
z = 3;

//Shorthand
let [x, y, z] = [1, 2, 3];
```

Assigning default value

```
let finalName;
let name = getName();
if(name !== null && name !== undefined && name !== '') {
    finalName = name;
} else {
    finalName = 'Bach'
}

// Shorthand
let finalName = getName() || 'Bach';
```

The ternary operator

```
//Long version
let points = 70;
let result;
if(marks >= 50){
    result = 'Pass';
}else{
    result = 'Fail';
}

//Shorthand
let points = 70;
let result = marks >= 50 ? 'Pass' : 'Fail';
```

Template Literals

```
// Long version
console.log('Hello ' + name +', it is ' + day);

//Shorthand
console.log(`Hello ${name}, it is ${day}`);
```

Let and Const

JavaScript LET and CONST is used to declare variables. Previously, variables were declared using the VAR keyword.

```
// variable declared using let
let name = 'Sara';
{
    // can be accessed only inside
    let name = 'Peter';

    console.log(name); // Peter
}
console.log(name); // Sara
```

```
// name declared with const cannot be changed
const name = 'Sara';
```

Let and Const

VAR vs LET vs CONST

	var	let	const
Stored in Global Scope	✓	✗	✗
Function Scope	✓	✓	✓
Block Scope	✗	✓	✓
Can Be Reassigned?	✓	✓	✗
Can Be Redeclared?	✓	✗	✗
Can Be Hoisted?	✓	✗	✗

Destructuring

The destructuring syntax makes it easier to assign values to a new variable. For example:

```
// before you would do something like this
const person = {
    name: 'Sara',
    age: 25,
    gender: 'female'
}

let name = person.name;
let age = person.age;
let gender = person.gender;

console.log(name); // Sara
console.log(age); // 25
console.log(gender); // female
```

Without ES6

Destructuring

Use ES6 Destructuring syntax, the above code can be written as:

```
const person = {
    name: 'Sara',
    age: 25,
    gender: 'female'
}

let { name, age, gender } = person;

console.log(name); // Sara
console.log(age); // 25
console.log(gender); // female
```

With ES6

Parameter Default

Default function parameters allow named parameters to be initialized with default values if no value or undefined is passed.

```
function sayHello(domain)
{
    // Tạo giá trị mặc định là freetuts.net
    domain = domain || 'freetuts.net';

    return domain;
}

// Sử dụng
console.log("KHÔNG truyền tham số: " + sayHello());
console.log("CÓ truyền tham số: " + sayHello('facebook.com'));
```

Before ES6

```
function createA() {
    return 5;
}

function add(a = createA(), b = a*2, c = b+3) {
    return a + b + c;
}

add() // 28 because 5 + (5*2) + ((5*2) + 3) = 5 + 10 + 1
add(2)// 13 because 2 + (2*2) + ((2*2) + 3) = 2 + 4 + 7 =
add(2,3)// 11 because 2 + 3 + (3+3) = 11
add(2,3,1)//6
```

ES6

Rest Parameter

The rest parameter syntax allows a function to accept an indefinite number of arguments as an array, providing a way to represent variadic functions in JavaScript.

```js
// es6 rest parameter
function add(...input){
    let sum = 0;
    for(let i of input){
        sum+=i;
    }
    return sum;
}

console.log(add(1,2));

console.log(add(2,4,6,8));
```

```
[Running] node "c:\Users\ozioma\Desktop\Learn\js\app.js"
3
20

[Done] exited with code=0 in 0.414 seconds
```

Spread Operator

Spread syntax (...) allows an iterable such as an array expression or string to be expanded in places where zero or more arguments (for function calls) or elements (for array literals) are expected, or an object expression to be expanded in places where zero or more key-value pairs (for object literals) are expected.

```
let arr1 = [1, 2, 3]
let arr2 = [...arr1]
console.log(arr2)
// [ 1, 2, 3 ]
```

```
let first = ["one", "two", "three"];
let second = [1, 2, 3]

let final = [...first, ...second];

console.log(final) //["one", "two", "three", 1, 2, 3]
```

```
const user = {
    firstname: 'Chris',
    lastname: 'Bongers'
}
const output = {...user, age: 31};
console.log(output);
// { firstname: 'Chris', lastname: 'Bongers', age: 31 }
```

Template String

Template literals are literals delimited with backtick (`) characters, allowing for multi-line strings, for string interpolation with embedded expressions, and for special constructs called tagged templates.

```
// Long version
console.log('Hello ' + name +', it is ' + day);

//Shorthand
console.log(`Hello ${name}, it is ${day}`);
```

Callback/Promise/Async Await

Callback

A callback is a function passed as an argument to another function

```
function hell(win) {
 // for listener purpose
 return function() {
   loadLink(win, REMOTE_SRC+'/assets/css/style.css', function() {
     loadLink(win, REMOTE_SRC+'/lib/async.js', function() {
       loadLink(win, REMOTE_SRC+'/lib/easyXDM.js', function() {
         loadLink(win, REMOTE_SRC+'/lib/json2.js', function() {
           loadLink(win, REMOTE_SRC+'/lib/underscode.min.js', function() {
             loadLink(win, REMOTE_SRC+'/lib/backbone.min.js', function() {
               loadLink(win, REMOTE_SRC+'/dev/base_dev.js', function() {
                 loadLink(win, REMOTE_SRC+'/assets/js/deps.js', function() {
                   loadLink(win, REMOTE_SRC+'/src/' + win.loader_path + '/loader.js', function() {
                     async.eachSeries(SCRIPTS, function(src, callback) {
                       loadScript(win, BASE_URL+src, callback);
                     });
                   });
                 });
               });
             });
           });
         });
       });
     });
   });
 };
}
```

Callback Hell

Promise

A JavaScript Promise object can be:

- Pending
- Fulfilled
- Rejected

How promise solve callback hell

```
getData(function(a) {
 getMoreData(function(b) {
  getMoreData(function(c) {
   getMoreData(function(d) {
    getMoreData(function(e) {
     // do something
    });
   });
  });
 });
})
```

```
getData()
.then(getMoreData)
.then(getMoreData)
.then(getMoreData)
.then(getMoreData)
.then((result) => {
// do something
})
.catch((error) => {
handleError(error);
});
```

How to use?

```
1   var promise = new Promise(function(resolve, reject){
2         resolve('Success');
3         // OR
4         reject('Error');
5     });
6
7   promise.then((success) => {
8           // Handle Success
9       }).cacth(err => {
10          //hanle error.
11      });
```

Async Await

async makes a function return a Promise.

await makes a function wait for a Promise.

```
(async function() {
    try {
        let response = await fetch('http://no-such-url');
    } catch(err) {
        console.log(err);
    }
})();
```

```
Const test = {
        Name:”dat”,
        Age: 24,
        Work : {
                A : “tester”,
                b:”REACT”,
                C : {
                        D : e,
                        F:g,
                        }
                }
}

Const { Name = ‘’, Work: { C: { D = 0 } = {}, C = {}} = {}} = test
```

TERRALOGIC

```
 TITLE>Thank You!</TITLE>
</HEAD>
<BODY>Thank You!</BODY>
</HTML>
```