ES6 and JS

ECMAScript 6

Javascript

Javascript ES6

var - let - const

Hoist or Hoisting in JavaScript

```
for(var i = 1; i < 10; i++) {
  //...
}
console.log(i);
```

Ans = ???

Local Scope & Global Scope

```
function sayHello() {
    var name = 'Chai';
}

console.log(name); // ReferenceError: name is not define
```

```
var name = 'Chai';

function sayHello() {

    for(var i =1; i < 5; i++) {
        console.log(name);
    }
}

sayHello(); // Chai
```

var name = variable in for or in function ?

```
function sayHello() {

    for(var i =1; i < 5; i++) {
        var name = 'Chai';
    }

    console.log(name);
}

sayHello(); // Name
```

Variable Declaration

```
var name = 'Chai';
```

Variable Assigment

```
name = 'Chai';
```

Hoist or Hoisting will move only **Variable Declaration**

Compare Variable Declaration & Assigment

```
function sayHello() {

    for(var i = 1; i < 5; i++) {
        var name = 'Chai';
    }

    console.log(name);
}

sayHello(); // Name
```

```
function sayHello() {
    var name;
    for(var i = 1; i < 5; i++) {
        name = 'Chai';
    }

    console.log(name);
}

sayHello(); // Name
```

```
for(var i = 1; i < 10; i++) {
  // ...
}
console.log(i);
```

```
var i;
for(i = 1; i < 10; i++) {
  // ...
}
console.log(i)
```

Ans = ???

var is function-scoped

```
function foo(isValid) {
    if(isValid) {
        var x = 1;
        return x;
    }

    return x;
}

// มีค่าเท่ากับ
function foo(isValid) {
    var x; // ดึดตัวเองมาอยู่ใกล้จุดประกาศฟังก์ชัน
    if(isValid) {
        x = 1;
        return x;
    }

    return x;
}
```

let & const is block-scoped

```
// พิมพ์ 2 ออกมาทั้งสองครั้ง
for(var i = 0; i < 2; i++) {
  // เนื่องจากในจังหวะที่ console.log ทำงาน loop ได้วนไปจนครบแล้ว
  // ทำให้ในขณะนั้นค่า i มีค่าเป็น 2
  // อย่าลืมว่า i ประกาศโดยใช้ var มันจึงผลักตัวเองออกจาก for
  // หรือพูดง่ายๆคือ i นั้นเป็นตัวแปรตัวเดียวทุกครั้งที่วนลูปก็เพิ่มค่าใส่ตัวแปรเดิม
  setTimeout(function() { console.log(i); }, 100);
}

// ผลลัพธ์ได้ 0 และ 1
// ใช้ let ประกาศตัวแปรทำให้ตัวแปรเป็น block-scoped
// พูดง่ายๆคือ i จะเกิดขึ้นทุกครั้งที่วนลูป
// และไม่ซ้ำกับ i ก่อนหน้า
// เนื่องจากเป็น block-scoped มันจึงมีช่วงชีวิตอยู่แค่ใน {}
for(let i = 0; i < 2; i++) {
  setTimeout(function() { console.log(i); }, 100);
}
```

Different between let & const

```
let a = 0
a = 1;

const PI = 3.14;
PI = 1; // "PI" is read-only เปลี่ยนค่าไม่ได้อีกแล้วนะ

// obj เก็บ address หรือที่อยู่ของ { a: 1 }
// เราเปลี่ยน address ไม่ได้ เช่น obj = {} แบบนี้คือเปลี่ยน memory address ทำไม่ได้
const obj = { a: 1 };
// แต่การเปลี่ยนค่าภายใน object ไม่ได้ทำให้ memory address เปลี่ยนไป
obj.a = 2;
console.log(obj); // { a: 2 }
```

Arrow Functions

Arrow Function

We used to declare the function by using the function keyword in ES2015.

We reduced the message to arrows or a fat arrow.

```
1  // ES5
2  function(arguments) {
3
4  }
5
6  // ES2015
7  (arguments) => {
8
9  }
10
```

```
function Dog() {
  this.color = 'white'

  setTimeout(function() {
    // this ตัวนี้หมายถึง this ใน context ของฟังก์ชันนี้
    // จึงไม่มีการพิมพ์อะไรออกไป เพราะในฟังก์ชันนี้ this ไม่มีค่าของ color
    console.log(this.color)
  }, 100)
}

new Dog()

// ถ้าต้องการให้พิมพ์ค่า color ออกมาต้องแก้ไขใหม่เป็น
function Dog() {
  this.color = 'white'
  let self = this

  setTimeout(function() {
    // เรียกผ่านตัวแปร self แทน
    console.log(self.color)
  }, 100)
}

// หรือใช้ arrow function ดังนี้
function Dog() {
  this.color = 'white'

  setTimeout(() => {
    // this ของ arrow function นี้จะหมายถึง
    // this ตัวบน
    console.log(this.color)
  }, 100)
}

```

Arrow function is not just a grammar change.

The arrow function also accesses **this** from the scope that covers it.

Case of Arrow Function if body of function is not covered by {}

A single statement assumes that the value is a return value from the function.

```
1  const fn = () => 3
2
3  console.log(fn()) // 3
4
```

```
1  const arr = [1, 2, 3]
2  // มีค่าเท่ากับ arr.map((x) => x * x)
3  arr.map(x => x * x)
4
```

Example Arrow Function

The function is represented by arrow

```
var greet = function(name, message) {
  return message + name;
}

var arrowGreet = (name, message) => {
  return message + name;
}
```

Example Arrow Function(2)

If in function not process or process in single line

```
var greet = function(name, message) {
  return message + name;
}

var arrowGreet = (name, message) => message + name;
```

```
var arrowGreet = message => message;

var square = x => x * x;
```

Example Arrow Function(3)

```
var person = {
  name: 'Luna',

  handleMessage: function(message, handler) {
    handler(message);
  },

  greet: function() {
    var _this = this;
    this.handleMessage('Hi', function(message) {
      console.warn(message + _this.name);
    })
  }
};

person.greet();
```

```
var person = {
  name: 'Luna',

  handleMessage: function(message, handler) {
    handler(message);
  },

  greet: function() {
    this.handleMessage('Hi', (message) => {
      console.warn(message + this.name);
    })
  }
};

person.greet();
```

Example Arrow Function(4)

```
var person = {
  name: 'Luna',

  handleMessage: function(message, handler) {
    handler(message);
  },

  greet: function() {
    this.handleMessage('Hi', (message) => console.warn(message + this.name))
  }
};

person.greet();
```

```
greet: function() {
  this.handleMessage('Hi', (message) => console.warn(message + this.name))
}
```

New Object(ES6)

```
1
2   const color = "red";
3
4   const age = 2;
5
6
7
8   function bark(){
9
10      console.log("hong");
11
12  }
13
14
15  //แบบที่ 1 : แบบปกติ
16
17  const dog = {color: color, age: age, bark: bark}
18
19
20
21  dog.bark();
22
23
24
```

```
23
24  //แบบที่ 2 : ลดรูปในกรณีที่ชื่อ Object เหมือนกัน
25  const dog = {color , age , bark}
26
27
28  //แบบที่ 3 : แต่เป็นแบบเก่า
29  const dog = {
30      color ,
31      age ,
32      bark: function(){
33          console.log("hong")
34      }
35  }
36
37  //แบบที่ 3 : inline function ไปไว้ใน object
38  const dog = {
39      color ,
40      age ,
41      bark(){
42          console.log("hong")
43      }
44  }
45
46
```

Spread & Rest

ES6

Spread Operator

Is ...

Spread Operator Number

```
//Example 1
const arr = [4,5,6];
const append = [1,2,3,arr];
console.log("Number 1 : " + append);
//Ans : [ 1, 2, 3, [ 4, 5, 6] ]
```

```
//Example 2
const arr2 = [4,5,6];
const append2 = [1,2,3, ...arr2];
console.log("Number 2 : " + append2);
//Ans : [ 1, 2, 3, 4, 5, 6 ]
```

```
//Example 3
const arr3 = [4,5,6];
const append3 = [...arr3 ,1,2,3 ];
console.log("Number 3 : " + append3);
//Ans : [ 1, 2, 3, 4, 5, 6 ]
```

Spread Operator String

```
//Example 1
const arr1 = ['a','b','c'];
const arr2 = ['d','e','f'];
arr1.push(arr2);
console.log("String 1 : " + arr1);
//Ans : [ 'a', 'b', 'c', [ 'd', 'e', 'f' ] ]


//Example 2
const arr3 = ['a','b','c'];
const arr4 = ['d','e','f'];
arr3.push(...arr4);
console.log("String 2 : " + arr3);
//Ans : [ 'a', 'b', 'c', 'd', 'e', 'f' ]
```

Slide = Copy

```
//Example 1 : แบบนี้มันเป็นแบบเก่า
const arr = [1,2,3];
const copy = arr.slice();
console.log(copy);
//Ans : 1,2,3

copy.push(4);
console.log(copy); // 1,2,3
console.log(arr); // 1,2,3,4
```

```
//Example 2 : แบบนี้ดีของใหม่
const arr = [1,2,3];
const copy = [...arr];
console.log(copy);
//Ans : 1,2,3

copy.push(4);
console.log(copy); // 1,2,3
console.log(arr); // 1,2,3,4
```

Spread into arguments

```
function add(x,y,z){
    return x + y + z;
}

const sum = add(1,2,3);
console.log(sum); //6
```

```
function add(x,y,z){
    return x + y + z;
}

const number = [1,2,3]
const sum = add(number);
console.log(sum); //6
```

```
function add(x,y,z){
    return x + y + z;
}

const number = [1,2,3]
const sum = add(...number);
console.log(sum); //6
```

Rest Operator

Is ...

Rest Operator = Inverse Spread Operator

```
function howManyArgs(...args){
    console.log(args.length);
    console.log(args);
}

howManyArgs() //0
howManyArgs(4) //1
howManyArgs(4,5) //2
howManyArgs(4,5,6,7) //4
howManyArgs(4,[5,6,7,]) //2
```

```
function multiply(multiplier,...array){
    console.log(multiplier);
    console.log(array);
    return array.map(e => multiplier * e) // e = element of array
    //.map เป็น function ของ array ที่จะ return array ตัวใหม่
}

const result = multiply(2,100,200,300);
console.log(result);
```

JS

ES6
destructuring

Destructuring Array

```
//Example 1 : แบบปกติ
const array = [1, 2]
const a = array[0]
const b = array[1]

console.log(a, b) // 1 2
```

```
//Example 2 : แบบ Destructuring
const array = [1, 2]
const [a, b] = array

console.log(a, b) // 1 2
```

Destructuring Array(2)

```
//Example 3 : Destructuring Array
const oneToFive = [1, 2, 3, 4, 5]
const [a, b] = oneToFive
console.log(a, b)
```

```
//Example 4 : Destructuring Array
const oneToFive = [1, 2, 3, 4, 5]
const [a, b, , d] = oneToFive
console.log(a, b ,d)
```

Destructuring Object

```
//Example 1 : แบบปกติ
const person = {first:'Jane' , last:'Done'}
const f = person.first
const l = person.last
console.log(f,l) // Jane Done
```

```
//Example 2 : แบบ Destructuring
const person = {first:'Jane' , last:'Done'}
const {first:f, last:l} = person
console.log(f,l) // Jane Done
```

Destructuring Object(2)

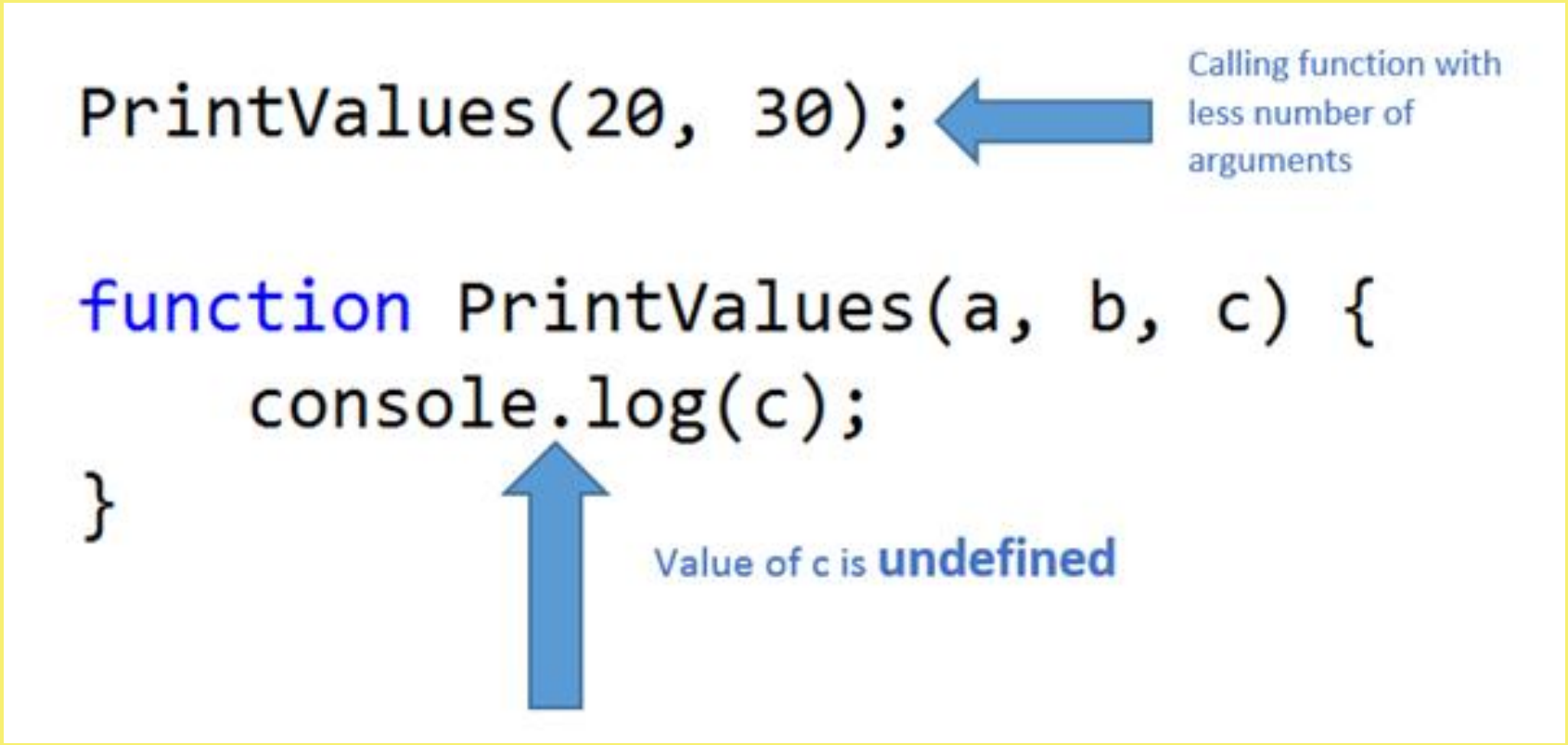
```
//Example 3 : Destructuring Object
const person = {
    first: 'Jane',
    last : 'Doe',
    email : 'jane@doe.com',
    birthday : {
        day : 20,
        month : 'Jan',
        year : 1990
    }
}

const {first: first, email:email} = person
console.log(first,email)
```

```
//Example 4 : Destructuring Object
const person = {
    first: 'Jane',
    last : 'Doe',
    email : 'jane@doe.com',
    birthday : {
        day : 20,
        month : 'Jan',
        year : 1990
    }
}

const {first, email} = person
console.log(first,email)
```

Destructuring Tip & Trick

```
// Tip & Trick : Rest & Spread + Destructuring
const array = [1, 2, 3, 4, 5]
const [a, b, ...rest] = array
console.log(rest); // [3,4,5]

const {a, ...rest} = {a:"1", b:"2", c:"3"}
console.log(rest)
```

Default Parameter



```
function howareyou(mood){

    if(! mood){
        mood = 'happy'
    }
    console.log(mood)
}

howareyou() //happy
```

```
function howareyou(mood = 'happy'){
    console.log(mood)
}

howareyou() //happy
howareyou('abc') //abc
```

Default Parameter(Array)

```
// Destructuring + Default Parameter[Array]

//Example 1
let [x,y] = [1,2]
console.log(x,y) // 1,2

//Example 2
let[x,y] = [1, ]
console.log(x,y) //1,undefind

//Example 3
let[x,y=3] = [1, ]
console.log(x,y) //1,3

//Example 4
let[x=2,y=3] = [1, ]
console.log(x,y) //1,3

//Example 5
let[x=2,y=3] = []
console.log(x,y) //2,3
```

Default Parameter(Object)

```
//Example 1
const {age:x , name:y} = {age:3,name:'Luna'}
console.log(x,y) // 3, Luna

//Example 2
const {age:x , name:y} = {name:'Luna'}
console.log(x,y) // undefind, Luna

//Example 3
const {age:x=3 , name:y} = {name:'Luna'}
console.log(x,y) // 3, Luna

//Example 4
const {age:x=3 , name:y='Nana'} = {}
console.log(x,y) // 3, Nana
```

Default Parameter Tip & Trick

```
//Example 1
const [{prop: x=3}] = [{prop:undefined}]
console.log(x) //3

//Example 2
const [{prop: x=3}] = [{}]
console.log(x) //3

//Example 3
const [{prop: x=3}] = []
console.log(x) //Error
//Destructure ไม่ได้จึงใส่ค่า Default ให้ไม่ได้

//Example 4
const [{prop: x=3} = {}] = []
console.log(x) //3
```

JS

ES6
classes

Constructor & Getter/Setter

```
class Rectangle{
    constructor(height,width){
        this.height=height;
        this.width=width;
    }

    greet(){
        console.log('Hi, my name is Lulu')
    }

    get color(){
        return this._color
    }

    set color(c){
        this._color = c
    }

    get dimension(){
        return this.width * this.height
    }

    static area(c){
        return c.width * c.height
    }
}
```

```
const r = new Rectangle(20,10)
console.log(r.height) //20
console.log(r.width) //10
r.greet() //Hi, my name is Lulu

r.color = 'red'
console.log(r.color) //red

console.log(r.dimension) //200

console.log(Rectangle.area(r)) //200
```

Inheritance

```
class Square extends Rectangle {
    constructor(width){
        super(width,width)
    }
}

const s = new Square(10)
console.log(s.dimension)
console.log(Square.area(s))
```

ES6
modules

Common js modules

```
function log(...msg){
    console.log(...msg)
}

function add(a,b){
    log('add',a,b)
    return a + b
}

function add1(a){
    return add(a,1)
}

log(add1(8))
// add 8 1
// 9
```

JS log.js

```
function log(...msg){
    console.log(...msg)
}

module.exports = log
```

JS index.js

```
const log = require('./log')

function add(a,b){
    log('add',a,b)
    return a + b
}

function add1(a){
    return add(a,1)
}

log(add1(8))
// add 8 1
// 9
```

Common es modules

```
function log(...msg){
    console.log(...msg)
}

function add(a,b){
    log('add',a,b)
    return a + b
}

function add1(a){
    return add(a,1)
}

log(add1(8))
// add 8 1
// 9
```

JS log.js

```
function log(...msg){
    console.log(...msg)
}

//module.exports = log //js modules
export default log //es module
```

JS index.js

```
//const log = require('./log') //js module
import log from './log' //es module

function add(a,b){
    log('add',a,b)
    return a + b
}

function add1(a){
    return add(a,1)
}

log(add1(8))
// add 8 1
// 9
```

Name export

```
function log(...msg){
    console.log(...msg)
}

function add(a,b){
    log('add',a,b)
    return a + b
}

function add1(a){
    return add(a,1)
}

log(add1(8))
// add 8 1
// 9
```

JS log.js

```
function log(...msg){
    console.log(...msg)
}

//module.exports = log //js modules
export default log //es module
```

JS math.js

```
function add(a,b){
    console.log('add',a,b)
    return a + b
}

function substract(a,b){
    console.log('add',a,b)
    return a - b
}

export {add,substract} //Name export
```

JS index.js

```
//const log = require('./log') //js module
import log from './log' //es module
import {add} from './math' //es module name export

function add1(a){
    return add(a,1)
}

log(add1(8))
// add 8 1
// 9
```

Function export

```
function log(...msg){
    console.log(...msg)
}

function add(a,b){
    log('add',a,b)
    return a + b
}

function add1(a){
    return add(a,1)
}

log(add1(8))
// add 8 1
// 9
```

JS log.js

```
function log(...msg){
    console.log(...msg)
}

//module.exports = log //js modules
export default log //es module
```

JS math.js

```
export function add(a,b){
    console.log('add',a,b)
    return a + b
}

export function substract(a,b){
    console.log('add',a,b)
    return a - b
}

//export {add,substract} //Name export
```

JS index.js

```
//const log = require('./log') //js module
import log from './log' //es module
import {add} from './math' //es module name export

function add1(a){
    return add(a,1)
}

log(add1(8))
// add 8 1
// 9
```

Code Lab