ES6 - NEW FEATURES

PHÓ NGHĨA VĂN

TÍNH NĂNG MỚI CỦA ES6

1. Từ khóa const và let
2. Arrow function
3. Export và import
4. OOP
5. Spread operator, rest params, default params
6. Template string
7. Vòng lặp với for...in... và for...of...

TỪ KHÓA CONST

- Dùng để khai báo hằng số (thay thế var)
- Chỉ có thể gán giá trị một lần (không thể re-assign)

```
const myName = "Hackagon";
console.log(myName);

myName = "something else";
console.log(myName);
```

```
"Hackagon"
"TypeError: Assignment to constant variable.
    at null.js:4:8
    at https://static.jsbin.com/js/prod/runner-4.1.4.min.js:1:13
    at https://static.jsbin.com/js/prod/runner-4.1.4.min.js:1:10
```

TỪ KHÓA LET

- Dùng để khai báo biến số (thay thế var)
- Có thể gán giá trị nhiều lần (re-assign)

PHÂN BIỆT LET VÀ VAR

Đầu tiên ta sẽ tìm hiểu về **function scope** và **block scope**:

- **Function scope**: còn gọi là **lexical scope**, phạm vi biến được khai báo bên trong một function. Biến bên trong function scope sẽ không thể truy cập từ bên ngoài. VD:

```
function getMyName() {
  var myName = 'Hackagon'
  console.log(myName);
}

getMyName();

console.log(myName);
```

```
"Hackagon"

"error"

"ReferenceError: myName is not defined
    at yajahoxigu.js:10:38
    at https://static.jsbin.com/js/prod/runner-4.1.4.min.js:1:13924
    at https://static.jsbin.com/js/prod/runner-4.1.4.min.js:1:10866"
```

PHÂN BIỆT LET VÀ VAR

Đầu tiên ta sẽ tìm hiểu về **function scope** và **block scope**:

- **Block scope**: phạm vi biến được khai báo bên trong { ... }. Biến bên trong block scope có thể được truy cập từ bên ngoài.

```
if(true){
  var myName = "Hackagon"
}

console.log(myName);
```

```
"Hackagon"
```

PHÂN BIỆT LET VÀ VAR

Phân việt let và var:

- Từ khóa **var**: biến được khai báo với từ khóa var bên trong một block scope **có thể** được truy cập từ bên ngoài. (như ví dụ trên)
- Từ khóa **let**: biến được khai báo với từ khóa let bên trong một block scope **không thể** được truy cập từ bên ngoài. VD:

```
if(true){
  let myName = "Hackagon"
}

console.log(myName);
```

```
"error"

"ReferenceError: myName is not defined
    at yajahoxigu.js:7:38
    at https://static.jsbin.com/js/prod/runner-4.1.4.min.js:1:13924
    at https://static.jsbin.com/js/prod/runner-4.1.4.min.js:1:10866"
```

PHÂN BIỆT LET VÀ VAR

```
if(true){
  var myName = "Hackagon"
  console.log("1. " + myName)
  if(true){
    var myName = "Nghĩa Văn"
    console.log("2. " + myName);
  }
}

console.log("3. " + myName);
```

```
"1. Hackagon"
"2. Nghĩa Văn"
"3. Nghĩa Văn"
```

```
if(true){
  let myName = "Hackagon"
  console.log("1. " + myName)
  if(true){
    let myName = "Nghĩa Văn"
    console.log("2. " + myName);
  }
}

console.log("3. " + myName);
```

```
"1. Hackagon"
"2. Nghĩa Văn"
"error"
"ReferenceError: myName is not defined
    at yajahoxigu.js:12:46
    at https://static.jsbin.com/js/prod/
    at https://static.jsbin.com/js/prod/
```

PHÂN BIỆT LET VÀ VAR

Ví dụ về sự khác nhau giữa **let** và **var** bên trong một callback function. Chú ý hàm **setTimeout** thuộc nodeAPI nên sẽ luôn được thực hiện cuối cùng trong chuỗi callstack. Khi đó, vòng lặp for đã chạy xong, và biến i có giá trị cuối cùng là 5

```
for(var i=0; i<5; i++){
  console.log("Outside callback: " + i);
  setTimeout(function(){
    console.log("Inside callback: " + i);
  }, 0)
}
```

```
"Outside callback: 0"
"Outside callback: 1"
"Outside callback: 2"
"Outside callback: 3"
"Outside callback: 4"
"Inside callback: 5"
"Inside callback: 5"
"Inside callback: 5"
"Inside callback: 5"
"Inside callback: 5"
```

PHÂN BIỆT LET VÀ VAR

Từ khóa **let** tỏa ra hiệu quả, khi được sử dụng trong callback function

```
for(let i=0; i<5; i++){
  console.log("Outside callback: " + i);
  setTimeout(function(){
    console.log("Inside callback: " + i);
  }, 0)
}
```

```
"Outside callback: 0"
"Outside callback: 1"
"Outside callback: 2"
"Outside callback: 3"
"Outside callback: 4"
"Inside callback: 0"
"Inside callback: 1"
"Inside callback: 2"
"Inside callback: 3"
"Inside callback: 4"
```

CONST, VAR, LET

Tóm tắt:

- Dùng **const**: để khai báo hằng số
- Dùng **var**: để khai báo biến số (hạn chế sử dụng)
- Dùng **let**: để khai báo biến số (dùng trong hầu hết các trường hợp)

ARROW FUNCTION

Arrow function được định nghĩa theo cú pháp:

(const) <tên hàm> = (<tham số>) => { ... }

```
function sum01(a, b){
   return a+b
}

console.log("Tổng của 1 và 2 là: " + sum01(1, 2))

sum02 = (a, b) => {
   return a+b
}

console.log("Tổng của 1 và 2 là: " + sum02(1, 2))
```

```
"Tổng của 1 và 2 là: 3"
"Tổng của 1 và 2 là: 3"
```

sum01(...)

sum02(...)

ARROW FUNCTION

Ưu điểm:

- Cách viết ngắn gọn:
  - Nhiều tham số
  - Một tham số
  - Không tham số

```
const sum = (a, b) => a + b;
console.log("Tổng của 1 và 2 là: " + sum(1, 2))

const sayHello = name => console.log("Hello " + name)
sayHello("hackagon");

const sayHello = () => console.log("Hello World");
sayHello();
```

ARROW FUNCTION

Ưu điểm:

- Từ khóa **this** trong function bình thường tham chiếu sẽ tham chiếu đến Object chứa nó.
-

```
let regularObj = function() {
  this.name = 'Hackagon';
  return {
    name: 'Nghĩa Văn',
    getName: function() {
      return this.name;
    }
  };
}
console.log('regularObj: ' + regularObj().getName());

// Arrow Function
let arrowObj = function() {
  this.name = 'Hackagon';
  return {
    name: 'Nghĩa Văn',
    getName: () => {
      return this.name;
    }
  };
}
console.log('arrowObj: ' + arrowObj().getName());
```

```
"regularObj: Nghĩa Văn"
```

```
"arrowObj: Hackagon"
```

EXPORT & IMPORT

- Thay cho module.exports và require trong ES5
- Khi dùng export clean tại utility.js thì tại app.js phải import đúng tên
- Khi dùng export **default** person tại person.js, thì tại app.js có thể import với tên bất kỳ.

person.js

```
const person = {
   name: 'Max'
}

export default person
```

utility.js

```
export const clean = () => { ... }

export const baseData = 10;
```

app.js

```
import person from './person.js'
import prs from './person.js'

import { baseData } from './utility.js'
import { clean } from './utility.js'
```

EXPORT & IMPORT

- Có thể đổi tên đối tượng được import như sau

```
import { BrowserRouter } from 'react-router-dom';

import { BrowserRouter as Router } from 'react-router-dom';
```

- Có thể import một lúc nhiều đối tượng như sau:

```
import * as actions from './actions/index.js';
import * as types from './constants/actionTypes.js';
```

OOP - LỚP ĐỐI TƯỢNG

- Tạo lớp đối tượng bằng **class**
- **constructor()** là phương thức khởi tạo sẽ được gọi khi đối tượng được khởi tạo

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

  getName(){
    console.log(this.name);
  }
}

const hackagon = new Person("Hackagon")
hackagon.getName();
```

OOP - KẾ THỪA

- Từ khóa **super()** dùng để gọi các thuộc tính và phương thức có lớp đối tượng cha.

```
class Human{
  constructor(){
    this.genger;
  }
  getGender(){
    console.log(this.gender)
  }
}

class Person extends Human{
  constructor(name, gender){
    super();
    this.name = name;
    this.gender = gender;
  }

  getName(){
    console.log(this.name);
  }
}

const hackagon = new Person("Hackagon", 23)
hackagon.getName();
hackagon.getGender();
```

OOP - ES6 vs ES7

Properties are like “variables attached to classes/ objects”

ES6

```
constructor () {
        this.myProperty = ' value'
}
```

ES7

```
myProperty = ' value'
```

Methods are like “functions attached to classes/ objects”

ES6

```
myMethod () { ... }
```

ES7

```
myMethod = () => { ... }
```

SPREAD OPERATOR

- Cú pháp **...** (dấu 3 chấm)
- Dùng để:
  - thêm một phần tử vào mảng, trả về mảng mới (array.push() trả về mảng cũ)
  - thêm một thuộc tính vào object, trả về object mới

```
let myThings = ["laptop", "phone", "keyboard"];
myThings.push("mouse");
console.log(myThings);

myStuffs = [...myThings, "desktop"];
console.log(myThings);
console.log(myStuffs);
```

```
["laptop", "phone", "keyboard", "mouse"]
["laptop", "phone", "keyboard", "mouse"]
["laptop", "phone", "keyboard", "mouse", "desktop"]
```

SPREAD OPERATOR

```
let myProfile = {
  name: "Hackagon",
  job: "lecturer"
}

myProfile.fullName = "Phó Nghĩa Văn"

myNewProfile = {
  ...myProfile, language: "English"
}

console.log(myProfile);
console.log(myNewProfile);
```

```
[object Object] {
  fullName: "Phó Nghĩa Văn",
  job: "lecturer",
  name: "Hackagon"
}
```

```
[object Object] {
  fullName: "Phó Nghĩa Văn",
  job: "lecturer",
  language: "English",
  name: "Hackagon"
}
```

REST PARAMS

- Sử dụng ... (3 chấm)
- Các tham số được khai báo, sẽ hợp lại thành một mảng

```
const printNumbers = (...numbers) => {
  numbers.forEach(num => console.log(num))
}

printNumbers(234,23,42,34,23,42,34,234);
```

```
234
23
42
34
23
42
34
234
```

DESTRUCTURING

- Array destructuring:

```
let [a, b] = [1, 2];
console.log(a + " and " + b)
```

```
"1 and 2"
```

- Object destructuring:

```
const myProfile = {
  name: "Hackagon",
  age: 23
}

let { name, age } = myProfile;

console.log("Name: " + name);
console.log("Age: " + age);
```

```
"Name: Hackagon"
"Age: 23"
```

DEFAULT PARAMS

Cho phép set giá trị mặc định cho tham số của hàm.

```
const getProfile = (name = "Hackagon", age = 23) => {
  return {
    name, age
  }
};

console.log(getProfile());
console.log(getProfile("Nghĩa Văn"));
console.log(getProfile(25));
console.log(getProfile(undefined, 25));
```

```
[object Object] {
  age: 23,
  name: "Hackagon"
}
[object Object] {
  age: 23,
  name: "Nghĩa Văn"
}
[object Object] {
  age: 23,
  name: 25
}
[object Object] {
  age: 25,
  name: "Hackagon"
}
```

TEMPLATE STRING

- Tạo một string vừa "tĩnh" vừa "động";
- String nằm trong dấu `...`
- Truyền giá trị động trong ${...}

```
let  name = "Hackagon";
let job = "fullstack developer";

console.log(`My name is ${name}, I am working as a ${job}`);
```

```
"My name is Hackagon, I am working as a fullstack developer"
```

For...in... và For...of...

- For...in... duyệt mảng theo index
- For...of... duyệt mảng theo từng phần tử

```
let myThings = ["laptop", "phone", "mouse", "keyboard"];
for(let index in myThings){
  console.log(myThings[index]);
}

for(let thing of myThings){
  console.log(thing);
}
```

```
"laptop"
"phone"
"mouse"
"keyboard"
```