





LẬP TRÌNH ECMASCRIPT

BIẾN VÀ THAM SỐ

- ⦿ Biết cách hoạt động của biến let và const
- ⦿ Hiểu được phạm vi của biến trong Javascript
- ⦿ Nắm được cách nối chuỗi mới với template string
- ⦿ Biết cách sử dụng Destructuring
- ⦿ Sử dụng Spread và ...Rest

-  Khai báo biến với Let và Const
-  Giới thiệu và làm việc với Template String
-  Tổng quan Destructuring
-  Tổng quan Spread và ...Rest

PHẦN 1: KHAI BÁO BIẾN VỚI LET VÀ CONST

- ❖ **let**: là từ khóa định nghĩa 1 biến có phạm vi truy cập trong 1 block – khối code.
- ❖ Cú pháp khai báo **let** cũng giống như cú pháp var, nhưng phạm vi của **let** chỉ trong dấu 1 cặp dấu {} bao quanh nó.
- ❖ Sử dụng **let** với ví dụ trước:

```
let a = 1;
if (a === 1) {
  let b = 2;
}
for (let c = 0; c < 3; c++) {
  // ...
}
function doSomething() {
  let d = 4;
}
console.log(a); // 1
console.log(b); // ReferenceError: b is not defined
console.log(c); // ReferenceError: c is not defined
console.log(d); // ReferenceError: d is not defined
```

KHAI BÁO LET TRONG VÒNG LẶP

❑ Khi hết vòng lặp giá trị của `i` sẽ bằng 5. Trong phần thân hàm, các giá trị được gọi sẽ lấy giá trị sau cùng của `i` nên `i = 5` trong các trường hợp.

```
var funcs = [];  
for (var i = 0; i < 5; i++) {  
  funcs.push(function(){  
    console.log(i);  
  });  
}  
funcs[3](); // sử dụng var sẽ có kết quả 5
```

❑ Với **let** trên mỗi lần lặp, vòng lặp tạo ra một biến mới `i`, điều này là nhờ phạm vi của `let` có hiệu lực trong khối `{}`

```
var funcs = [];  
for (let i = 0; i < 5; i++) {  
  funcs.push(function(){  
    console.log(i);  
  });  
}  
funcs[3](); // sử dụng let sẽ có kết quả 3
```

Lưu ý: let cũng hoạt động theo cách tương tự với các vòng lặp for..in và for..of.

❑ Ví dụ với vòng lặp for...in

```
var funcs = [],
    object = {a: true, b: true, c: true};
for (let key in object) {
    funcs.push(function() {
        console.log(key);
    });
}
funcs.forEach(function(func) {
    func(); // outputs "a", then "b", then "c"
});
```

- ❑ Mỗi lần chạy vòng lặp, biến **key** sẽ được khởi tạo mới do đó mỗi hàm sẽ có giá trị của biến **key** riêng. Kết quả là mỗi hàm xuất ra một giá trị khác nhau.
- ❑ Nếu sử dụng **var** để khai báo cho key, tất cả các hàm sẽ hiển thị ra c.

- ❑ Việc truy cập biến khai báo **let** sớm hơn so với khai báo của nó sẽ gây ra lỗi, trong khi với khai báo **var**, thứ tự không quan trọng.

```
console.log(a);    // undefined
console.log(b);    // ReferenceError!
var a;
let b;
```


KHAI BÁO BIẾN VỚI CONST

- ❑ **const**: là từ khóa định nghĩa 1 biến sẽ là hằng số. **Biến const** lưu trữ giá trị không thể thay đổi được trong suốt vòng đời của biến.
- ❑ **const** là một biến chỉ đọc sau khi trị ban đầu của nó được đặt

```
const a = 2;  
console.log(a); // 2  
a = 3; // báo lỗi Uncaught TypeError: Assignment to constant variable
```

- ❑ Giá trị của các biến được khai báo bằng **const** không thể thay đổi sau khi đã đặt. Vì lý do này, mọi biến const phải được khởi tạo và gán giá trị khi khai báo:

```
const maxItems = 30; // Đúng cú pháp  
const name; // Báo lỗi SyntaxError: Missing initializer in const declaration
```

KHAI BÁO CONST TRONG VÒNG LẶP

❑ Hằng số cũng có thể sử dụng trong vòng lặp

```
var funcs = [],
    object = {a: true, b: true, c: true};
// Không gây ra lỗi
for (const key in object) {
    funcs.push(function() {
        console.log(key);
    });
}
funcs.forEach(function(func) {
    func(); // kết quả "a", then "b", then "c"
});
```

Lưu ý: Giá trị của **key** không thể thay đổi bên trong vòng lặp.

- ❑ Hằng số không phải bất biến trong mọi trường hợp. Nếu giá trị phức tạp, chẳng hạn như một đối tượng hoặc mảng, giá trị vẫn có thể được thay đổi. Ví dụ:

```
const a = [1,2,3];  
a.push(4);  
console.log(a); // [1,2,3,4]
```

```
a = 42; // SyntaxError
```

PHẦN 2: GIỚI THIỆU TEMPLATE STRING

- ❑ Là một tính năng mới của ES6 giúp làm việc với chuỗi và các chuỗi dễ dàng hơn so với phiên bản ES5
- ❑ Bao bọc văn bản bằng dấu **`basktick`** (thay vì dấu ngoặc kép hoặc dấu nháy đơn)
- ❑ Ngoài ra template string hỗ trợ những tính năng:
 - ❖ Có thể sử dụng cho nhiều dòng
 - ❖ Có thể sử dụng biểu thức (expression), biến(variable) trong chuỗi
 - ❖ Sử dụng các ký tự đặc biệt an toàn hơn

TEMPLATE STRING LÀ GÌ?



❑ Cú pháp cơ bản

```
let message = `Hello world!`;  
console.log(message);           // "Hello world!"  
console.log(typeof message);    // "string"
```

❑ Nếu muốn sử dụng ký tự đặc biệt chỉ cần thêm dấu \ phía trước

```
let message = `\\`Hello\\` world!`;  
  
console.log(message);           // "`Hello` world!"
```

- ❑ Trong ES5, khi sử dụng dấu ngoặc kép hoặc dấu nháy đơn, các chuỗi phải được chứa hoàn toàn trên một dòng.
- ❑ Cách giải quyết cho thiếu sót này là sử dụng **join** và **\n**

```
var message = [  
    "Multiline ",  
    "string"  
].join("\n");
```

```
let message = "Multiline \n" +  
    "string";
```

MULTIPLE STRING

- ❑ Trong ES6, các ký tự mẫu làm cho các chuỗi nhiều dòng trở nên dễ dàng vì không có cú pháp đặc biệt. Chỉ cần bao gồm xuống dòng nơi bạn muốn.

```
let message = `Multiline
string`;
console.log(message);    // "Multiline
                        //  string"
```

- ❑ Tất cả các khoảng trắng bên trong dấu `` đều là được chấp nhận của chuỗi, vì vậy cần lưu ý khi canh lề.

```
let message = `Multiline
                string`;
console.log(message); // "Multiline
                      // string"
```


BIỂU THỨC TRONG TEMPLATE STRING

- ❑ Template string cho phép sử dụng biểu thức trong chuỗi, thông qua cú pháp `\${...}`
- ❑ Ví dụ:

```
var name = "Max";  
console.log(`Hello, ${name}!`);  
// => "Hello, Max!"
```

```
var a = 10;  
var b = 10;  
console.log(`The number of JS MVC frameworks is  
|         |      ${2 * (a + b)} and not ${10 * (a + b)}.`);  
// => Kết quả: The number of JS frameworks is 40 and not 200.
```

Lưu ý: Việc cố gắng sử dụng một biến chưa khai báo trong template string sẽ gây ra lỗi

BIỂU THỨC TRONG TEMPLATE STRING

- ❑ `${...}` có thể làm việc tốt với bất kì biểu thức nào bao gồm: hàm, phương thức
- ❑ Ví dụ:

```
const fn = () => {  
  return "Chào mừng các bạn đến với ES6.";  
};
```

```
console.log(`Hi, ${fn()}`);  
// => Hi, |Chào mừng các bạn đến với ES6.
```

```
var user = {  
  name: 'Max'  
};  
console.log(`Chúc mừng sinh nhật, ${user.name.toUpperCase()}.`);  
// => "Chúc mừng sinh nhật, MAX";
```

❑ Ví dụ nâng cao về xây dựng cấu trúc HTML

```
var li = (obj) => `<li><a href="${obj.url}">${obj.label}</a></li>`;
var ul = (arr) => `<ul>${arr.map((obj) => li(obj)).join('\n')}</ul>`;
var arr = [
  {url: "http://www.twitter.com", label: "Twitter"},
  {url: "http://www.linkedin.com", label: "Linked In"},
  {url: "http://www.facebook.com", label: "Facebook"}
];
document.getElementById('list').innerHTML = ul(arr);
```

❑ Kết quả

```
<ul>
  <li><a href="http://www.twitter.com">Twitter</a></li>
  <li><a href="http://www.linkedin.com">Linked In</a></li>
  <li><a href="http://www.facebook.com">Facebook</a></li>
</ul>
```

PHẦN 3: TỔNG QUAN DESTRUCTURING

DESTRUCTURING LÀ GÌ?

- ❑ Destructuring là một cú pháp cho phép bạn gán các thuộc tính của một Object hoặc một Array.
- ❑ Có hai loại Destructuring: Objects Destructuring và Arrays Destructuring.
- ❑ Ví dụ **Destructoring** cho phép tạo nhiều biến dựa trên thuộc tính cùng lúc

```
var obj = {first: 'Jane', last: 'Leo'};
```

```
var {first: a, last: b} = obj;  
// a = 'Jane', b = 'Leo'
```

```
var [x, y] = ['Jane', 'Leo'];  
// x = 'Jane', y = 'Leo'
```

Thêm một ví dụ về **Destructuring**

```
var foo = {bar: 'pony', baz: 3};  
var {bar: a, baz: b} = foo;  
console.log(a)  
// 'pony'  
console.log(b)  
// 3
```

Nếu sử dụng **ES5** (Phiên bản JS trước đây)

```
var foo = {bar: 'pony', baz: 3};  
var a = foo.bar;  
var b = foo.baz;
```

❑ **Destructuring** cho phép truy cập thuộc tính xuống sâu nếu muốn.

ES6

```
var foo = {  
  bar: {  
    deep: 'pony',  
    smile: 'lol'  
  }  
};  
var {bar: {deep, smile: sure}} = foo;  
console.log(deep);  
// 'pony'  
console.log(sure);  
// 'lol'
```

ES5

```
var foo = {  
  bar: {  
    deep: 'pony',  
    smile: 'lol'  
  }  
};  
var deep = foo.bar.deep;  
var sure = foo.bar.smile;
```

- ❑ Ngoài ra **Destructring** cũng có thể áp dụng cho mảng bằng cách sử dụng dấu ngoặc vuông cho phần khai báo biến:

```
const [a, b, c] = [1, 2, 3];
```

```
console.log("A is " + a + ", B is " + b + ", C is " + c);  
// A is 1, B is 2, C is 3
```


KẾT HỢP OBJECT VÀ ARRAYS

- ❑ Chúng ta cũng có thể kết hợp các đối tượng và mảng với nhau và sử dụng các thuộc tính của chúng.

```
var mixed = {  
  one: 1, two: 2, values: [3, 4, 5]  
};  
var {one: a, two: b, values: [c, , e]} = mixed;  
console.log(a, b, c, e);  
// 1 2 3 5
```

- ❑ Tương tự với giá trị trả về của một hàm:

```
function mixed() {  
  return {  
    one: 1, two: 2, values: [3, 4, 5]  
  };  
}  
var { one: a, two: b, values: [c, , e] } = mixed();
```

❑ Bạn cũng có thể sử dụng destructuring trong một vòng lặp (es6 loop)

❑ Ví dụ:

```
let arr = [{name: 'Jane', age: 41}, {name: 'John', age: 40}];
for (var {name, age} of arr) {
  console.log(name, age);
}
// Jane 41
// John 40
```

PHẦN 3: TỔNG QUAN SPREAD VÀ ...REST

SPREAD OPERATOR LÀ GÌ?

- ❖ ES6 giới thiệu một toán tử ... mới thường được gọi là toán tử **spread** hoặc **rest**, tùy thuộc vào vị trí hoặc cách nó được sử dụng.
- ❖ **Spread Operator** cho phép chuyển đổi một chuỗi thành nhiều argument (trong trường hợp gọi với hàm) hoặc thành nhiều phần tử (cho array).

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

```
const numbers = [1, 2, 3];
```

```
console.log(sum(...numbers));  
// expected output: 6
```

```
console.log(sum.apply(null, numbers));  
// expected output: 6
```

- ❑ Trước đây, khi muốn tạo một mảng mới, gộp mảng chúng ta thường sử dụng `push()`, `splice()`, `concat()` Nhưng với toán tử Spread Operator mọi việc được xử lý tiện lợi hơn rất nhiều.
- ❑ Ví dụ:

ES5

```
let parts = ['four', 'five'];  
let numbers = ['one', 'two', 'three'];  
console.log(numbers.concat(parts));
```

// Kết quả : ["one","two","three","four","five"]

ES6

```
let parts = ['four', 'five'];  
let numbers = ['one', 'two', 'three'];  
console.log([...numbers, ...parts]);
```

// Kết quả : ["one","two","three","four","five"]

- ❑ Toán tử Spread Operator cũng có thể áp dụng cho Object
- ❑ Nó sao chép các thuộc tính được khai báo từ một đối tượng được cung cấp vào một đối tượng mới.

```
const obj1 = { foo: 'bar', x: 42 };  
const obj2 = { foo: 'baz', y: 13 };
```

```
const clonedObj = { ...obj1 };  
// Object { foo: "bar", x: 42 }
```

```
const mergedObj = { ...obj1, ...obj2 };  
// Object { foo: "baz", x: 42, y: 13 }
```

REST PARAMETER LÀ GÌ?

❑ **Rest parameter** (tham số còn lại) : là tính năng Javascript cho phép bạn có thể khai báo một hàm với số lượng tham số không xác định.

❑ Cú pháp

```
function f(a, b, ...thamso) {  
    // ...  
}
```

❑ Ví dụ

```
function myFun(a, b, ...manyMoreArgs) {  
    console.log("a", a)  
    console.log("b", b)  
    console.log("manyMoreArgs", manyMoreArgs)  
}
```

```
myFun("one", "two", "three", "four", "five", "six")
```

```
// Kết quả :
```

```
// a, one
```

```
// b, two
```

```
// manyMoreArgs, [three, four, five, six]
```

REST PARAMETER LÀ GÌ?

❑ Ví dụ Tính tổng 2 số

```
function caculate(action,x,y)
{
    let result=0;
    switch (action) {
        case '+':
            result=x+y;
            break;
        case '-':
            result=x-y;
            break;
    }
    return result;
}
console.log(caculate('+',2,3));
```

KẾT QUẢ
???

REST PARAMETER LÀ GÌ?

❑ Ví dụ Tính tổng 2 số

```
function caculate(action,...values)
{
    let result=0;
    switch (action) {
        case '+':
            for(let value of values)
                result+=value;
            break;
        case '-':
            for(let value of values)
                result-=value;
            break;
    }
    return result;
}
console.log(caculate('+',10,3,4,8,6));
```

KẾT QUẢ BAO
NHIÊU CHO PHÉP
"_"
???

- ☑ Khai báo biến với Let và Const
- ☑ Giới thiệu và làm việc với Template String
- ☑ Tổng quan Destructuring
- ☑ Tổng quan Spread và ...Rest

A rustic, handmade tag made of light brown cardboard is placed on a sandy surface. The tag has a small hole on the left side, through which a dark, thin string is threaded. To the right of the tag, a single white daisy with a bright yellow center is in sharp focus. In the background, two more similar daisies are visible but are out of focus. The entire scene is set against a backdrop of fine, light-colored sand.

Thank
you!