VIETNAM AUSTRALIA Vocational College

Hàm, phạm vi hàm, IIFEs, hàm mũi tên, mảng

Mentor: Nguyễn Bá Minh Đạo

Nội dung:

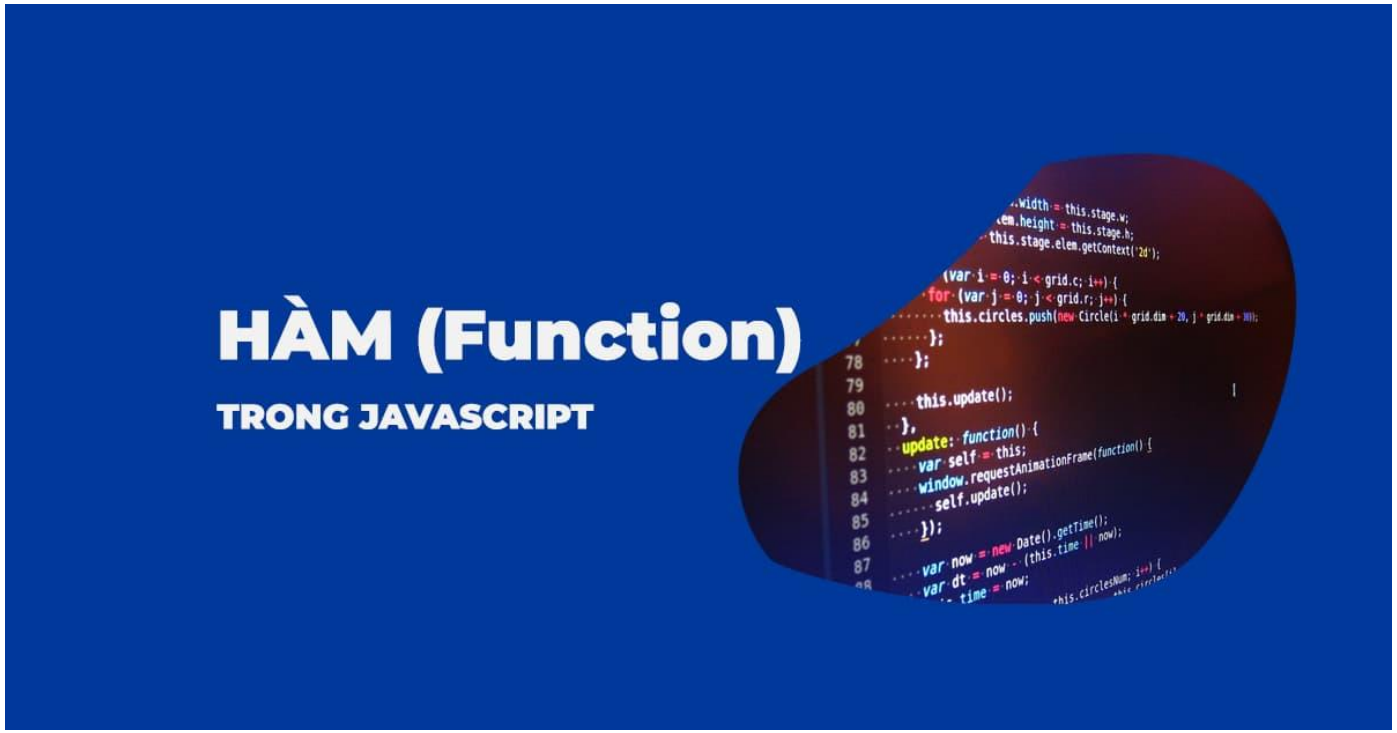
1. Hàm, thao tác hàm
2. Phạm vi hàm
3. Hàm biểu thức IIFEs
4. Hàm mũi tên
5. Mảng, thao tác mảng

Hàm, thao tác hàm

❑ **Tóm tắt khái niệm**:

➢ **Hàm** (function) là **một sự chia nhỏ** của **chương trình**.

➢ **Hàm cho phép xác định** một **khối mã** (block scope), **đặt tên hàm** và **có thể** được **thực thi nhiều lần** (ở mỗi lần cần dùng)



Hàm, thao tác hàm

❑ Tóm tắt khái niệm:

➢ Để **khai báo hàm** ta phải **sử dụng từ khóa function**.

➢ **Cú pháp**:

```
function <function-name> () {
    // block scope
    // code to be executed
};
```

➢ **Ví dụ**:

function.js

```
1   function showMessage() {
2       console.log("Hello VUSers!");
3   };
```

Hàm, thao tác hàm

❑ Gọi hàm:

- ➢ Khai **báo hàm xong, hàm** của chúng ta vẫn **chưa thể hoạt động.**
- ➢ **Để hàm hoạt động**, ta cần **phải gọi hàm** ở vị trí cần sử dụng hàm.
- ➢ **Cú pháp**:

```
<function-name> ();
// another code
// after function executed
```

- ➢ **Ví dụ**:

call-function.js

```
1   function showMessage() {
2       console.log("Hello VUSers!");
3   };
4   showMessage();
```

Hàm, thao tác hàm

- **Biến ngoài hàm** (outer variables):
  - **Outer variables** (global variable – **biến toàn cục**) là **biến được khai báo ngoài hàm**, có thể **được truy xuất trong** và **ngoài hàm**.
  - **Ví dụ**:

outer-variables.js

```
let userName = "FastTracker";

function showMessage() {
    let message = "Hello, " + userName;
    console.log(message);
};

showMessage();
```

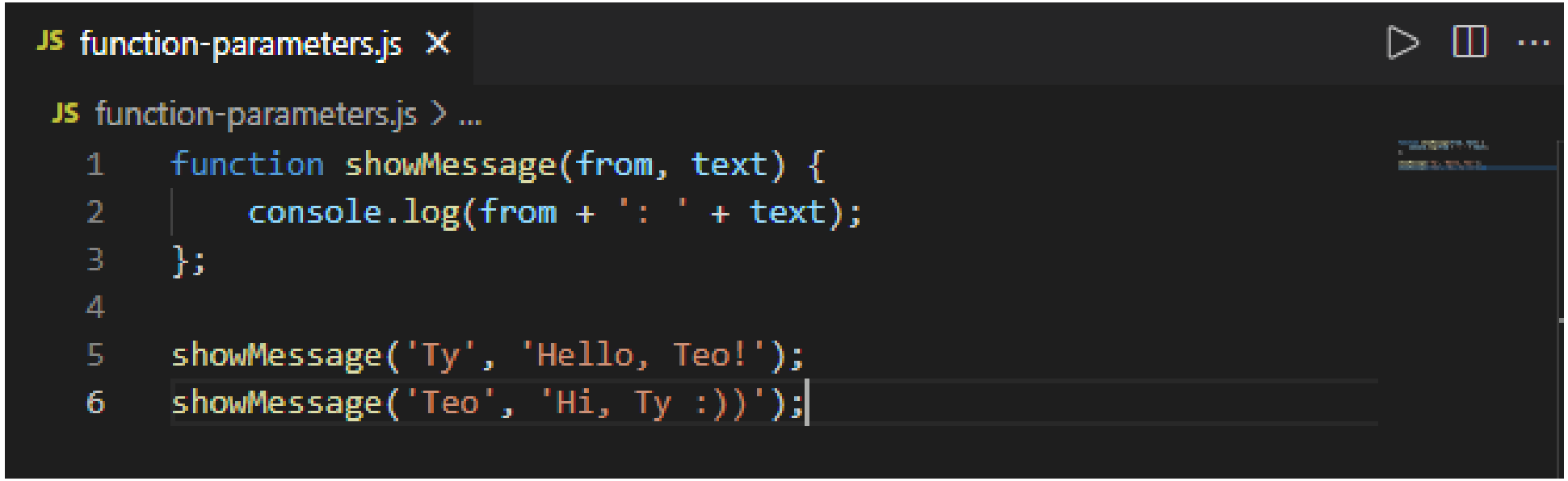
Hàm, thao tác hàm

- **Tham số hàm** (function parameters):
  - **Tham số hàm** là **các thông số đầu vào của một hàm**.
  - Chúng ta **có thể sử dụng một** hoặc **nhiều tham số ở 1 hàm**.
  - **Cú pháp**:

```
function <function-name> (para01, para02,...,.) {
        // block scope
        // code to be executed
};
```

Hàm, thao tác hàm

- **Tham số hàm** (function parameters):
  - **Ví dụ:**



  - Trong đó, ta có **2 tham số hàm**:
    - from
    - text

Hàm, thao tác hàm

- **Tham số mặc định** (default parameters):
  - Ta cũng có thể **đặt giá trị mặc định cho tham số** của **hàm**.

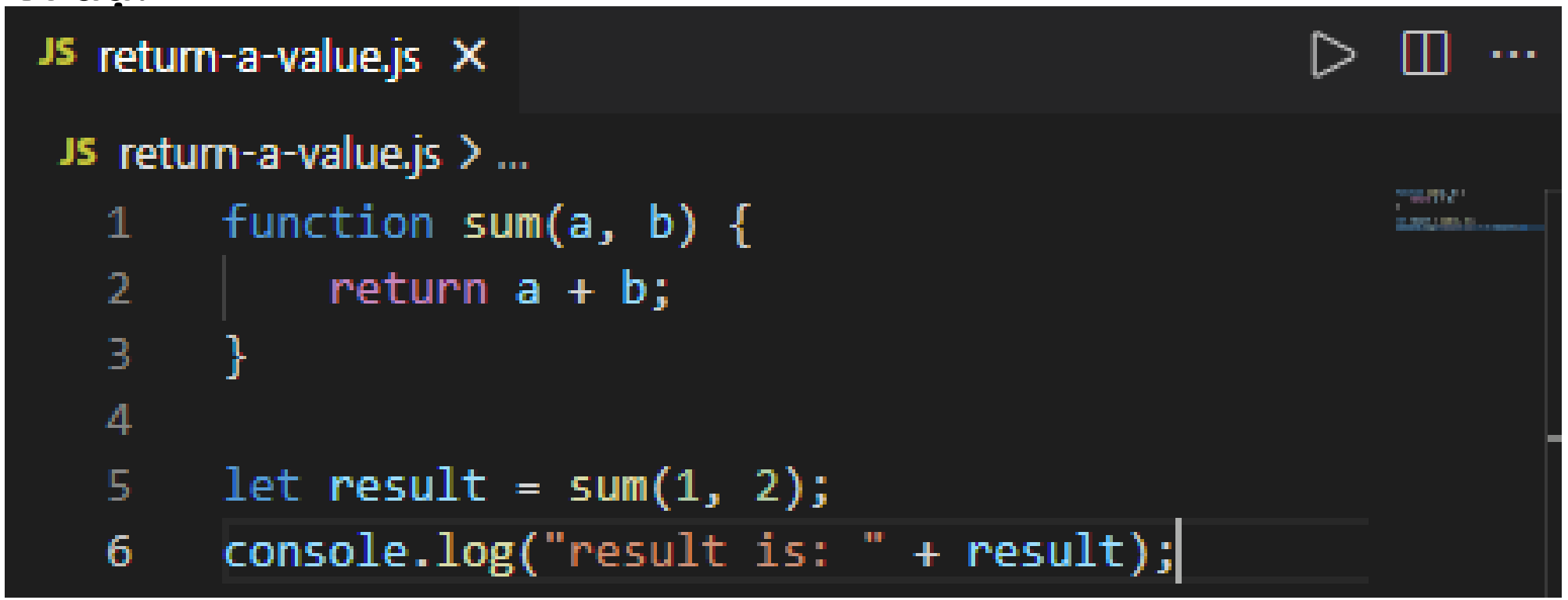
default-parameters.js

```
function showMessage(from, text = "Hello VUsers!") {
    console.log(from + ': ' + text);
};

showMessage('Teo');
```

  - Trong đó, ta có 1 **tham số text được thiết lập giá trị mặc định**:
    - text = “Hello VUSers!”

Hàm, thao tác hàm

- **Hàm trả về giá trị** (return a value):
  - **Hàm** cũng **có thể trả về** cho chúng ta **một giá trị**.
  - **Ví dụ**:



Hàm, thao tác hàm

- **Biểu thức hàm** (function expression):
  - **Hàm** cũng **có thể được chỉ định như một biến**.
  - **Hàm được gọi ẩn trong 1 biến** thì sẽ **không được gọi ở ngoài biến đó** (tức là **phải dùng thông qua biến đó**).

function-expression-01.js

```
1   var add = function sum(val1, val2) {
2       return val1 + val2;
3   };
4
5   var result = add(10, 20);
6   console.log("result: " + result);
```

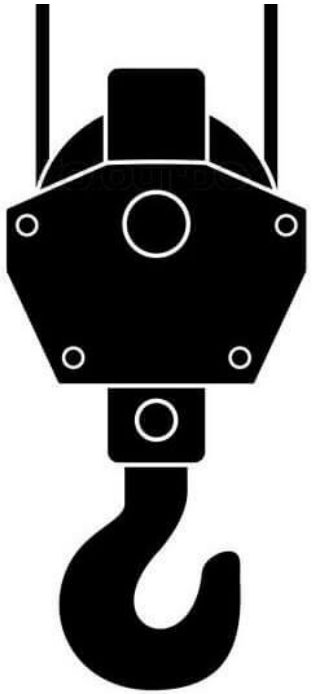
function-expression-02.js

```
1   let add = function sum(val1, val2) {
2       return val1 + val2;
3   };
4
5   var result1 = add(10, 20);
6   var result2 = sum(10, 20); //sum is not defined
7   console.log("result1: " + result1);
8   console.log("result2: " + result2);
```

Phạm vi hàm

❑ Cẩu lên (Hoisting):

➢ **Hoisting** trong JavaScript về cơ bản là **chỉ việc trước khi bất kỳ đoạn code nào được thực thi** thì **tất cả những khai báo biến, định nghĩa hàm sẽ được "kéo" lên – "cẩu" lên** (di chuyển) **lên trên đầu của scope hiện tại**.

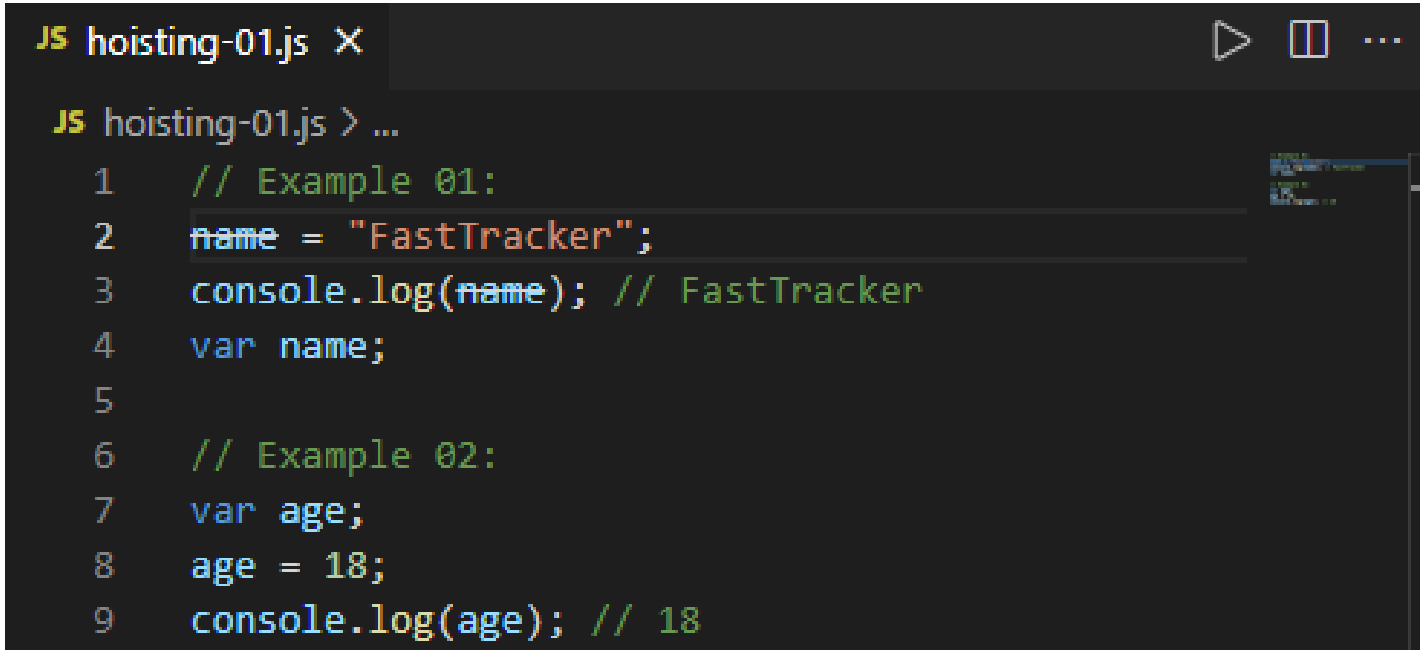


JAVASCRIPT
HOISTING

Phạm vi hàm

❑ Cẩu biến lên (Hoisting Variables):

➢ Việc **hoisting** sẽ được thực hiện và code của chúng ta sẽ chạy bình thường, việc **khai báo biến name/age** sẽ **được chuyển lên đầu**.

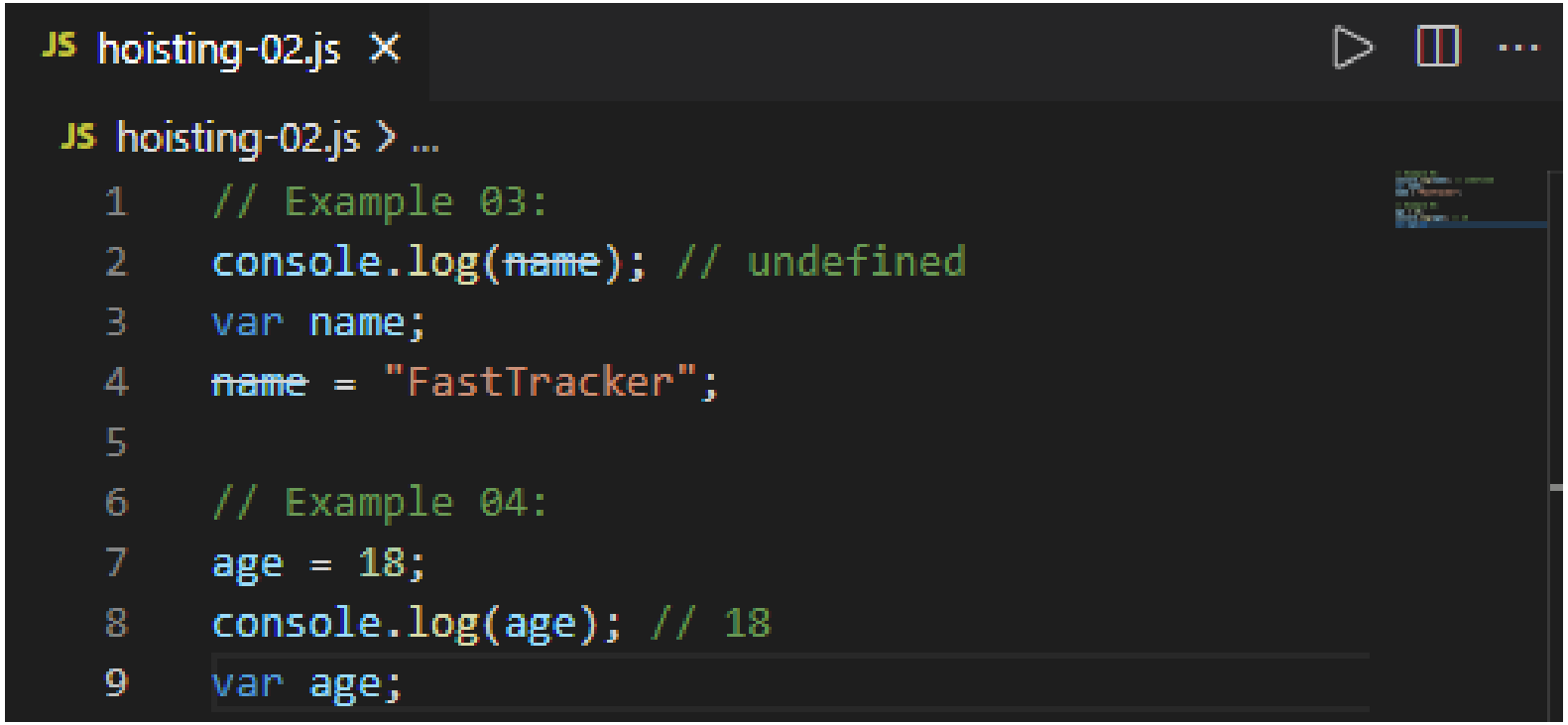


➔ Cả **2 kết quả này** cho thấy chúng **tương đương nhau**, **biến name** và **biến age đã được khai báo**.

Phạm vi hàm

❑ Cẩu biến lên (Hoisting Variables):

➢ Nếu ta chưa gán giá trị cho biến **name** thì kết quả là **undefined**. Ngược lại ví dụ 4 cho thấy kết quả ra đúng với giá trị biến **age** được gán bằng **18**. => **phần gán giá trị của biến không được Hoisting**



Phạm vi hàm

- **Cẩu hàm lên (Hoisting Functions):**
  - **Ví dụ 5:** đã khai báo biến **name**, chưa khởi tạo giá trị -> **undefined**.
  - **Ví dụ 6:** đã khai báo và gán giá trị biến **name** -> '**FastTrackers**'

hoisting-03.js

```
// Example 05:
function say() {
    console.log(name);
    var name;
    name = 'FastTrackers';
}

say(); // undefined
```

hoisting-04.js

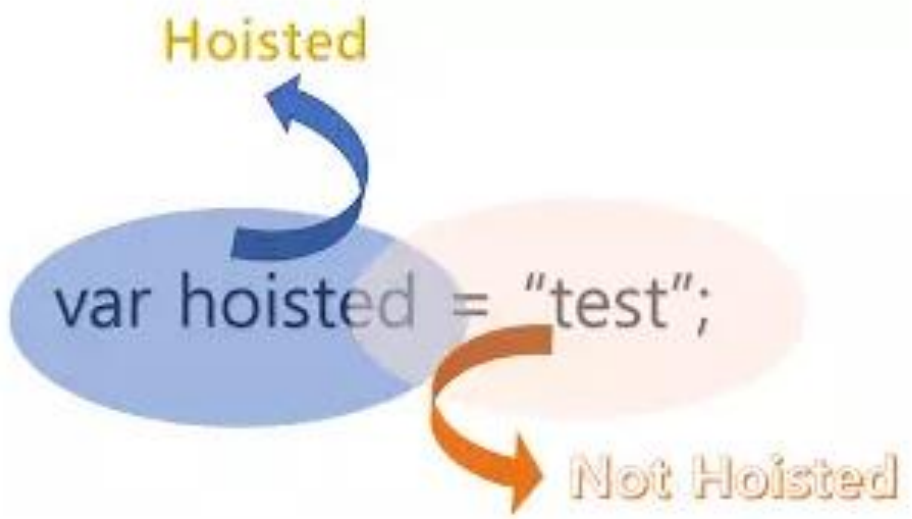
```
// Example 06:
function say() {
    var name;
    name = 'FastTrackers';
    console.log(name);
}

say(); // FastTrackers
```

Phạm vi hàm

❑ **Cẩu hàm lên (Hoisting Functions):**

➢ Qua 2 ví dụ (5, 6) ta thấyrằng:

▪ Đối với **Hoisting Function**, **hoisting** là **việc đưa các khai báo biến lên phần đầu tiên của function**, nhưng **còn phần gán giá trị của biến**, thì sẽ **không được di chuyển lên phần đầu tiên của function**



Hàm biểu thức IIEFs

- Để **một hàm** được **thực thi**, ta **cần** lời **gọi hàm**.
- Nhưng ta có **1 cách khác** để **thực thi 1 hàm** mà **không cần** lời **gọi hàm**, nó thường được gọi là **immediately invoked function expressions** (**IIFEs**)
- **IIFEs** là **hàm biểu thức thực hiện ngay lập tức**.
- **Ví dụ**:

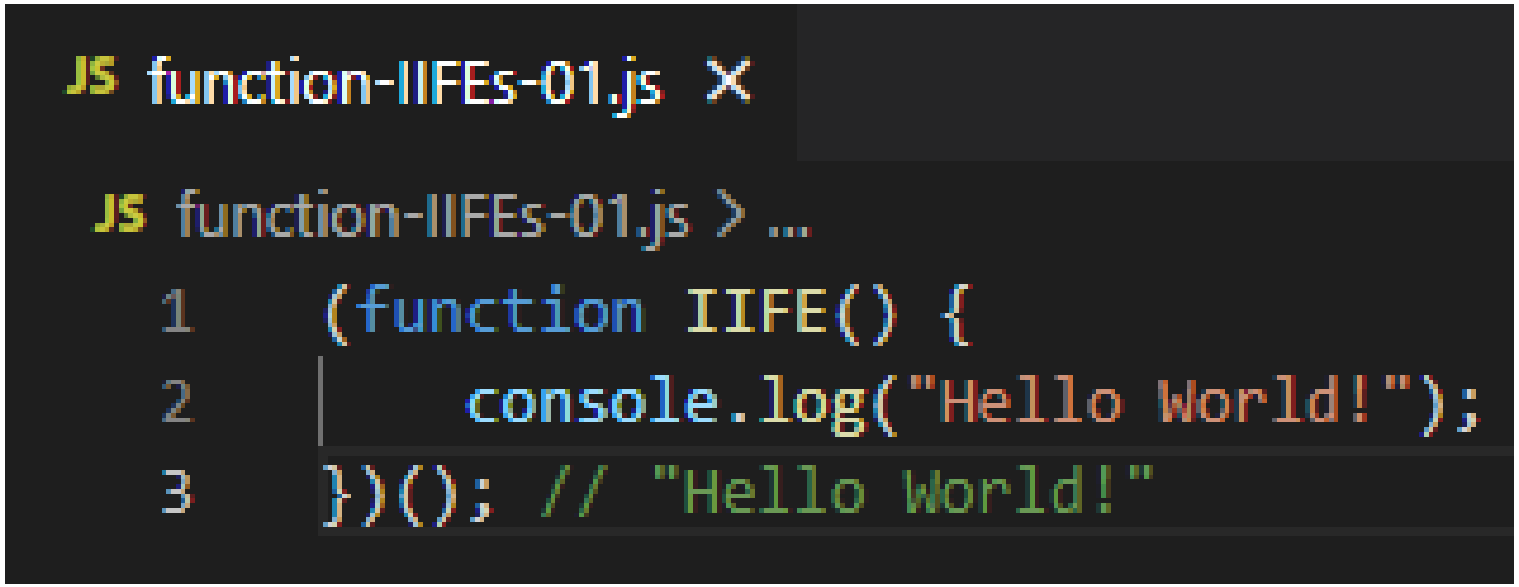
JS function-IIFEs-01.js

JS function-IIFEs-01.js > ...

```
1   (function IIFE() {
2       console.log("Hello World!");
3   })(); // "Hello World!"
```

Hàm biểu thức IIEFs

❑ **Ví dụ**:



❑ Dấu **()** cuối cùng của biểu thức **})();** là chính xác **cái gì sẽ thực thi tức thì trước nó**.

❑ **()** là **cần thiết** tương tự 1 hàm thông thường trước khi thực thi nó bằng **();** trong cả hai trường hợp, function đại diện thực thi ngay tức thì sau dấu **()**. **Ví dụ**: function hello() {..}; hello**()**; -> thực thi hàm hello đi!

Hàm biểu thức IIEFs

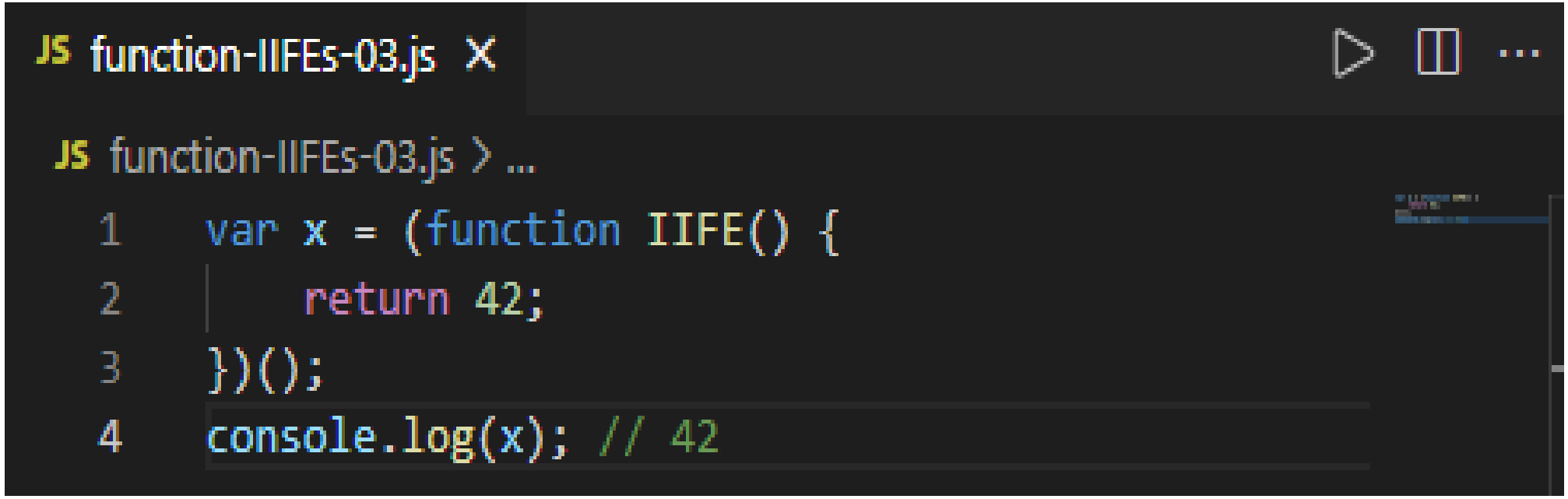
- Bởi vì **IIFE** chỉ là **một function** và **function thì tạo phạm vi biến**, **sử dụng IIFE theo cách này** thường là **để khai báo biến không ảnh hưởng đến code bên ngoài IIFE**.
- **Ví dụ**:

function-IIFEs-02.js

```
var a = 42;
(function IIFE() {
    var a = 10;
    console.log(a); // 10
})();
console.log(a); // 42
```

Hàm biểu thức IIEFs

- Ngoài ra, **IIFEs** còn có thể **trả về kết quả** là **giá trị** của **một biến** mà biến này **được gán cho giá trị trả về** của **một hàm**.
- **Ví dụ**:



- Giá trị **42** được **return** từ **IIFE** – thực thi function được đặt tên theo **x**.

Hàm mũi tên

❑ **Hàm mũi tên** (arrow function):

➢ Hàm mũi tên (**arrow function**) là **một cú pháp mới dùng để viết các hàm** trong **JavaScript.**

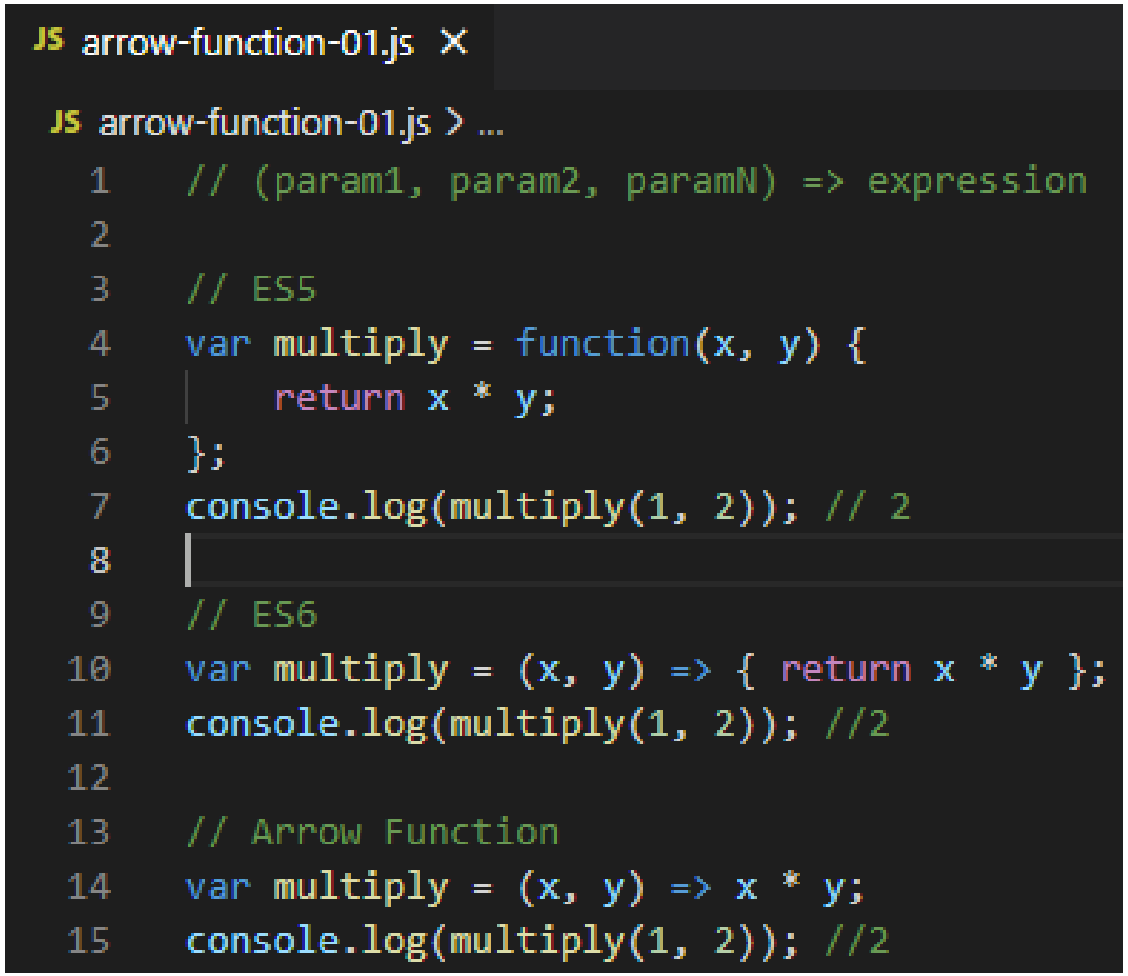
➢ Hàm mũi tên (**arrow function**) **giúp tiết kiệm thời gian phát triển** và **đơn giản hóa phạm vi function** (function scopes)

➢ Hàm mũi tên (**arrow function**) - còn gọi là **"fat arrow"** là **có cú pháp ngắn gọn để viết function.** Sử dụng ký tự **=>, trông giống một mũi tên "béo"**.

```
() => {}
```

Hàm mũi tên

- **Hàm mũi tên** (arrow function):
  - Sử dụng **arrow function** có **nhiều tham số**:
    - Các ví dụ bên sử dụng **ES5, ES6, arrow function** đều ra cùng kết quả.
    - Nhưng **cú pháp** của **arrow function** là **ngắn gọn nhất**.



Hàm mũi tên

- **Hàm mũi tên** (arrow function):
  - Sử dụng **arrow function không có tham số**:
    - Dấu ngoặc đơn là không bắt buộc khi không có tham số.
  - Sử dụng **arrow function có một tham số**:
    - Dấu ngoặc đơn là không bắt buộc khi chỉ có một tham số.

arrow-function-03.js

```
1  // ES5
2  var hello = function sayHello() {
3      console.log("Hello World");
4  }
5
6  // ES6
7  var hello = () => { console.log("Hello World"); }
8
9  hello(); // Hello World
```

arrow-function-02.js

```
1  //ES5
2  var phraseSplitterEs5 = function phraseSplitter(phrase) {
3      return phrase.split(' ');
4  };
5
6  //ES6
7  var phraseSplitterEs6 = phrase => phrase.split(" ");
8
9  console.log(phraseSplitterEs6("Hello World")); // ["Hello", "World"]
```

Mảng, thao tác mảng

- **Mảng** là **một object giữ các giá trị** (của bất kỳ kiểu nào) **có vị trí theo chỉ số** (không phải theo một khóa/thuộc tính được đặt tên). **Ví dụ**:
- JavaScript **dùng 0** như là **chỉ mục** của **giá trị đầu tiên trong mảng**.
- Quan sát **giá trị** của **arr** một cách **trực quan** như sau:

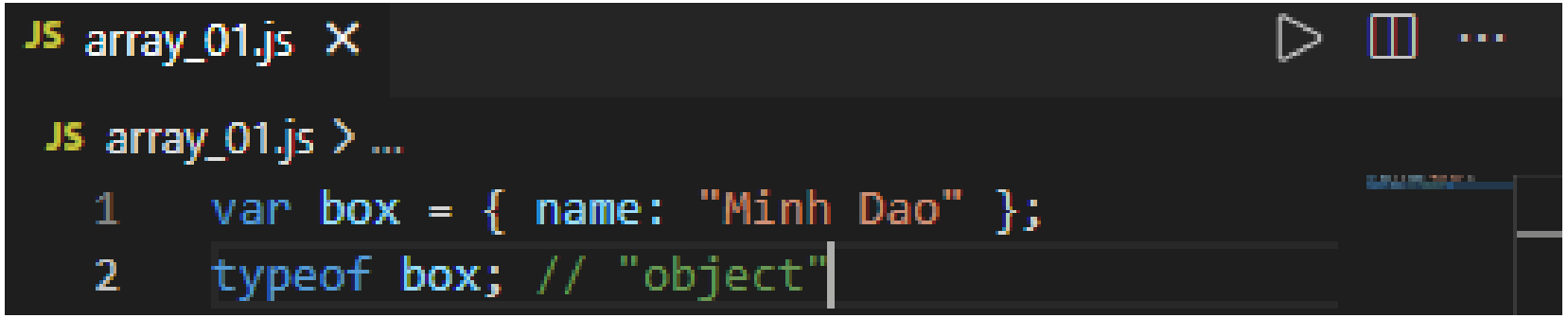
arr

0: "hello world"	1: 18	2: true

```
JS array_02.js
JS array_02.js > ...
1   var arr = [
2       "hello world",
3       18,
4       true
5   ];
6   arr[0]; // "hello world"
7   arr[1]; // 18
8   arr[2]; // true
9   arr.length; // 3
10  typeof arr; // "object"
```

Mảng, thao tác mảng

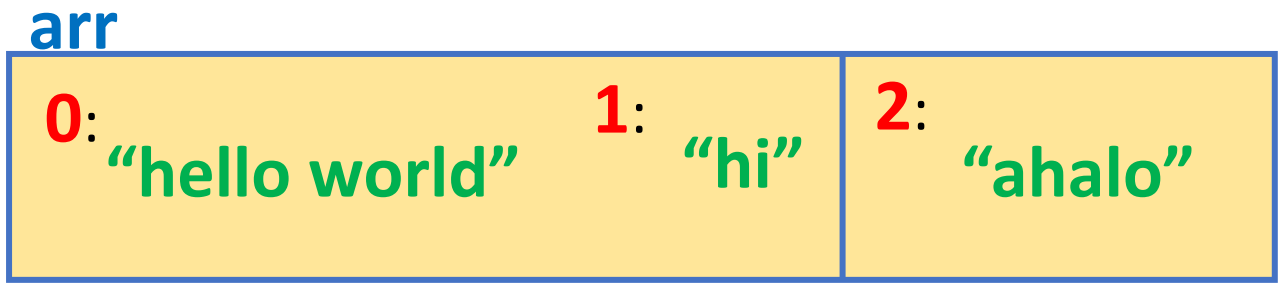
- **Mảng** là **một object đặc biệt** (**như typeof** đã **ngụ ý** ở ví dụ dưới đây)



- **Mảng** có thể **có thuộc tính** (thuộc tính **length** cũng được **tự động cập nhật**
- Bạn **có thể sử dụng mảng như một object bình thường**.
- Bạn cũng **có thể sử dụng một object nhưng chỉ cho các thuộc tính số** (0, 1, ..) **tương tự** như **một mảng**.
  - **Không nên** vì quy định **nên sử dụng đúng loại tương ứng**.
  - **Nên sử dụng array**, **object** phù hợp.

Mảng, thao tác mảng

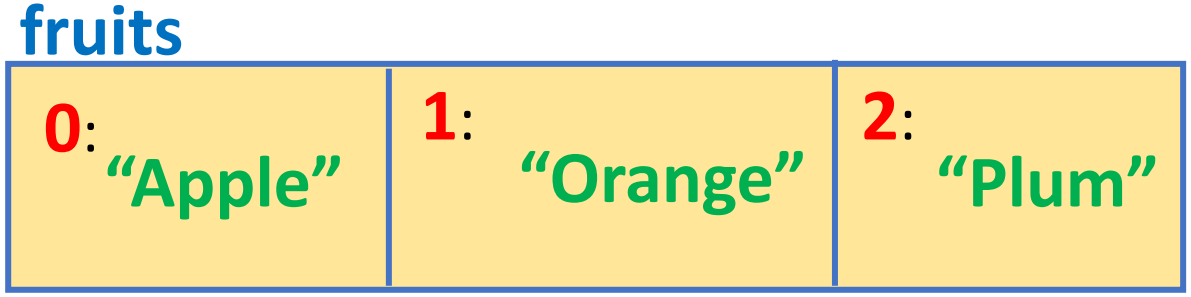
- Về bản chất, **mảng** là **một object đặc biệt**. Nhưng chúng ta **thường sử dụng mảng với một số đặc tính riêng biệt** để hỗ trợ các vấn đề lập trình cụ thể liên quan với mảng.
- Một số **đặc tính mảng** cần lưu ý như sau:
  - Mảng là **một dãy các phần tử** có **cùng cấu trúc** và **lưu trữ liên tiếp trong bộ nhớ**. **Các phần tử** trong **mảng** có **cùng kiểu** và **cùng tên**.
  - **Việc truy xuất** đến một **phần tử** trong **mảng** được **thực hiện thông qua biến chỉ số**.



Mảng, thao tác mảng

- **Tạo 1 mảng 1 chiều rỗng**:
  - **Cách 1**: `let arr = new Array[];`
  - **Cách 2**: `let arr = [];`
- **Tạo 1 mảng 1 chiều có 3 phần tử**:

  `let fruits = ["Apple", "Orange", "Plum"];`
- Hình ảnh **minh họa mảng fruits**:



Mảng, thao tác mảng

- Hình ảnh **minh họa mảng fruits**:

fruits

0: "Apple"	**1**: "Orange"	**2**: "Plum"

- **Truy xuất phần tử** trong **mảng fruits**:
  - **fruits[0];**
  - **fruits[1];**
  - **fruits[2];**

array_03.js

```
let fruits = ["Apple", "Orange", "Plum"];
console.log(fruits[0]); // "Apple"
console.log(fruits[1]); // "Orange"
console.log(fruits[2]); // "Plum"
```

Mảng, thao tác mảng

- Hình ảnh **minh họa mảng fruits**:

fruits

0: "Apple"	1: "Orange"	2: "Plum"

- **Thay đổi giá trị phần tử** trong **mảng fruits**:
  - **fruits[2] = "Pear";**

array_04.js

```
1   let fruits = ["Apple", "Orange", "Plum"];
2   fruits[2] = "Pear";
3   console.log(fruits[2]); // "Pear"
```

Mảng, thao tác mảng

- Hình ảnh **minh họa mảng fruits**:

fruits

0: "Apple"	1: "Orange"	2: "Plum"

- **Thêm 1 phần tử mới** trong **mảng fruits**:
  - **fruits[3] = "Lemon";**

array_05.js

```
let fruits = ["Apple", "Orange", "Plum"];
fruits[3] = "Lemon";
console.log(fruits[3]); // "Lemon"
```

Mảng, thao tác mảng

- Các **thao tác** (**nhập/xuất/gọi hàm**) với **mảng**:

array_06.html × array_06.js ×

array_06.js > ...

```
1   // Input array:
2   var arr = [];
3   var n;
4
5   function input() {
6       n = prompt("Enter number n: ");
7       for (var i = 0; i < n; i++) {
8           arr[i] = prompt("Enter a number arr[" + i + "]: ");
9       }
10  }
11
12  // Output array:
13  function output() {
14      for (var i = 0; i < n; i++) {
15          document.write("a[" + i + "]: " + arr[i] + "<br>");
16      }
17  }
18
19  // Call function:
20  input(arr, n);
21  output(arr, n);
```

Mảng, thao tác mảng

- Các **thao tác** (**nhập/xuất/gọi hàm**) với **mảng**:
  - **Copy mảng** trong **JavaScript**: **array.slice()**
  - **array.slice()** **trả về bản copy của mảng vào một mảng mới** từ đầu đến cuối. **Mảng ban đầu sẽ không bị sửa đổi**.

array_07.js

```
animals = ["ant", "bison", "camel", "duck", "elephant"];

console.log(animals.slice(0));
// ["ant", "bison", "camel", "duck", "elephant"]

console.log(animals.slice(2));
// ["camel", "duck", "elephant"]

console.log(animals.slice(2, 4));
// ["camel", "duck"]

console.log(animals.slice(1, 5));
// ["ant", "bison", "camel", "duck", "elephant"]
```

Tổng kết:

- ❑ Hàm, thao tác hàm
- ❑ Phạm vi hàm
- ❑ Hàm biểu thức IIFEs
- ❑ Hàm mũi tên
- ❑ Mảng, thao tác mảng