VIETNAM AUSTRALIA Vocational College

LẬP TRÌNH JAVASCRIPT CƠ BẢN

HÀM, PHẠM VI HÀM

Nội dung:

1. Lệnh break & continue
2. Hàm (Function), cách dùng hàm
3. Viết chương trình sử dụng hàm
4. Phạm vi hàm (Function Scopes)
5. Hàm biểu thức IIFEs
6. Hàm mũi tên (Arrow Function)

Lệnh break và continue

❑ **Lệnh break**:

➢ Lệnh **break** được sử dụng **để thoát ra khỏi cấu trúc lặp (for, while, do..while)** và **cấu trúc rẽ nhánh (switch..case)** mà

➢Lệnh **break không** cần **dùng điều kiện kết thúc vòng lặp.**

➢ **Khi** có **nhiều vòng lặp lồng nhau**, **break thoát ra** khỏi **vòng lặp bên trong khối lệnh lặp** chứa nó.

JS break.js

JS break.js > ...

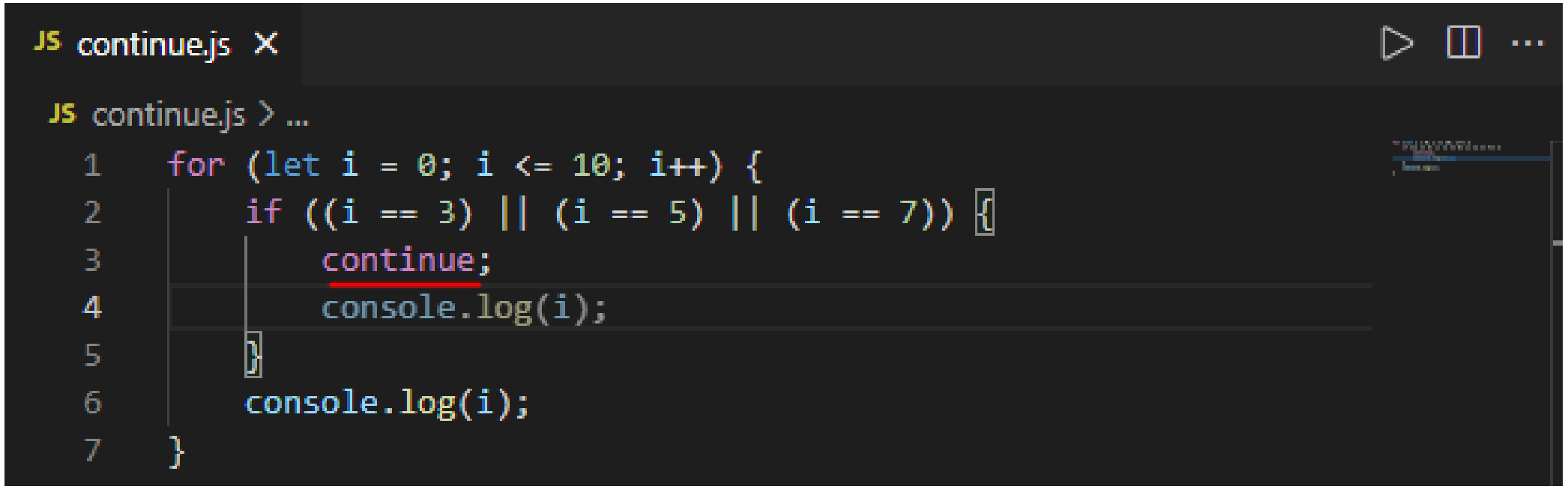
```
1   for (let i = 0; i <= 10; i++) {
2       if ((i == 3) || (i == 5) || (i == 7)) {
3           break;
4           console.log(i);
5       }
6       console.log(i);
7   }
```

Lệnh break và continue

❑ **Lệnh continue:**

➢ Lệnh **continue** được sử dụng **để bắt đầu** một **vòng mới của cấu trúc lặp chứa nó**.

➢ Lệnh **continue** thường được sử dụng **bên trong thân của vòng lặp for**.



Hàm (function), cách dùng 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)

➢ 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 (function), cách dùng 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 (function), cách dùng 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 (function), cách dùng 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 (function), cách dùng hàm

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

function-parameters.js

```
function showMessage(from, text) {
    console.log(from + ': ' + text);
};

showMessage('Ty', 'Hello, Teo!');
showMessage('Teo', 'Hi, Ty :))');
```

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

Hàm (function), cách dùng hàm

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

default-parameters.js

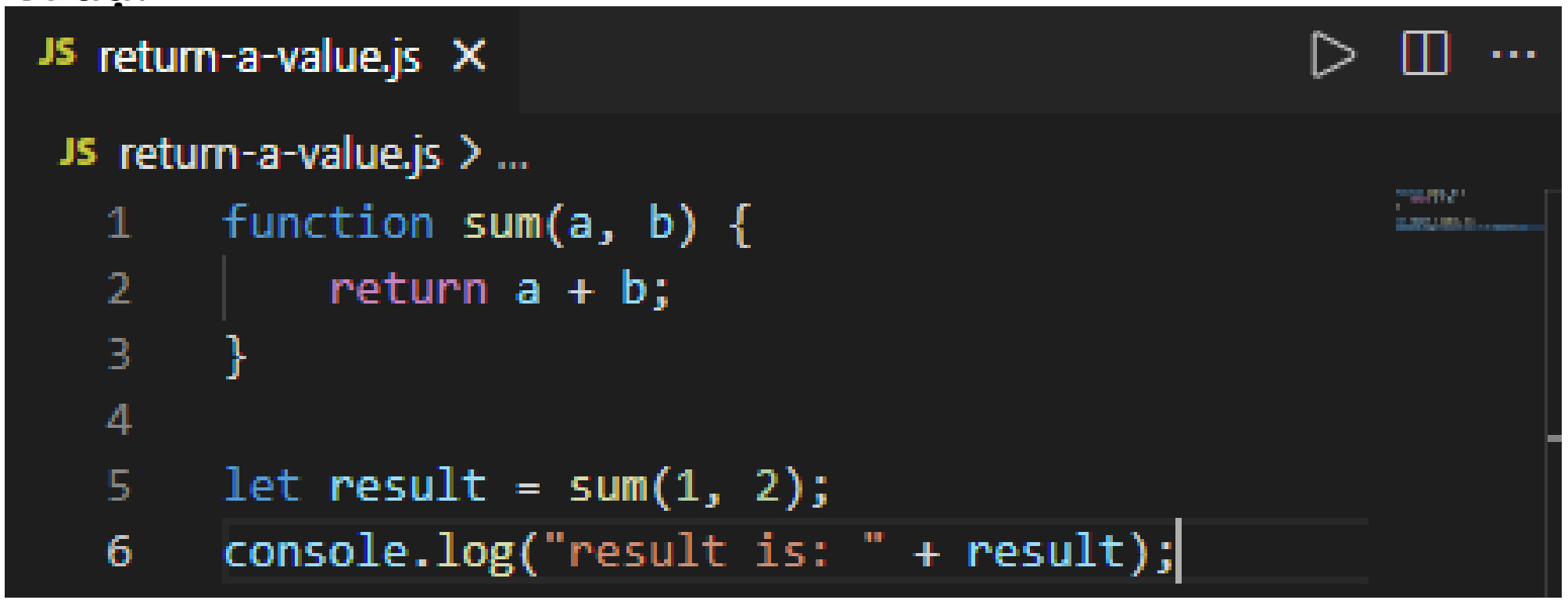
```
function showMessage(from, text = "Hello VUsers!") {
    console.log(from + ': ' + text);
};

showMessage('Teo');
```

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

Hàm (function), cách dùng hàm

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



Hàm (function), cách dùng hàm

- **Biểu thức hàm** (function expression):
  - **Hàm** cũng **có thể được chỉ định như một biến**.
  - **Ví dụ**:

function-expression-01.js

```
var add = function sum(val1, val2) {
    return val1 + val2;
};

var result = add(10, 20);
console.log("result: " + result);
```

Hàm (function), cách dùng hàm

- **Biểu thức hàm** (function expression):
  - **Hàm được gọi ẩn trong 1 biến** thì sẽ **không được gọi ở ngoài biến đó** (tức là **phải dùng thông qua biến đó**).
  - **Ví dụ**:

function-expression-02.js

```
let add = function sum(val1, val2) {
    return val1 + val2;
};

var result1 = add(10, 20);
var result2 = sum(10, 20); //sum is not defined
console.log("result1: " + result1);
console.log("result2: " + result2);
```

Viết chương trình sử dụng hàm

❑ Viết **một chương trình** để **tính toán tổng giá điện thoại bạn đã mua** với **một số yêu cầu chi tiết** trong **1 tình huống cụ thể** như sau:

➢ Bạn sẽ tiếp tục mua điện thoại (dùng **vòng lặp**) cho đến khi bạn hết tiền trong tài khoản (mỗi cái điện thoại giá $99.99).

➢ Bạn đồng thời cũng sẽ mua phụ kiện ($9.99) cho mỗi cái điện thoại miễn là số tiền dưới ngưỡng chi tiêu ($200).

➢ Sau khi bạn tính toán tổng tiền, thêm thuế (8%), sau đó in ra tổng cuối cùng với định dạng hoàn chỉnh.

➢ Cuối cùng, kiểm tra số tiền đối với số dư trong tài khoản ngân hàng để xem bạn có đủ khả năng hay không. (không đủ -> xuất ra không đủ tiền mua).

Viết chương trình sử dụng hàm

❑ **Hướng dẫn** (nâng cấp dùng **prompt** nếu muốn):

➢ **Tạo các biến hằng**: **ngưỡng chi tiêu** ($200), **tỉ lệ thuế** (8%), **giá điện thoại** ($99.99), **giá phụ kiện** ($9.99**), tài khoản ngân hàng** ($303.91), **tổng tiền** ($0 -> sẽ thay đổi sau khi tính toán)

➢ **Viết 2 hàm**: **tính thuế** (tổng tiền * tỉ lệ thuế), **định dạng tổng tiền** (cộng ký tự $ vào trước tổng tiền, tổng tiền làm tròn 2 số thập phân)

➢ **Thực hiện quá trình mua hàng**: **mua điện thoại**, **mua phụ kiện** (nếu tổng tiền hiện tại chưa vượt quá ngưỡng chi tiêu dự định ban đầu)

➢ **Tính lại tổng tiền sau khi mua hàng** xong (cộng thêm thuế)

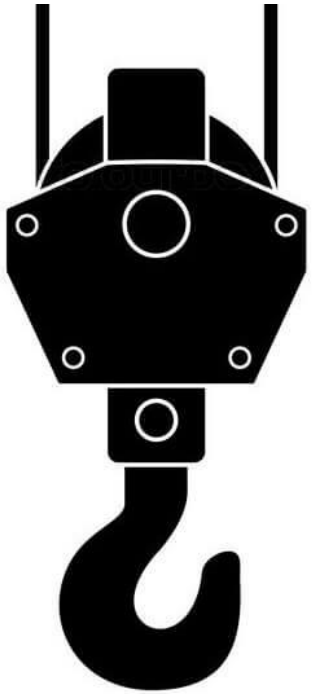
➢ **In** ra màn hình cửa sổ lệnh **tổng tiền phải trả sau khi định dạng**.

➢ **Kiểm tra tổng tiền** hiện tại **so với tài khoản ngân hàng**. (in ra lố tiền rồi)

Phạm vi hàm (Function Scopes)

❑ Cẩu lên (Hoisting):

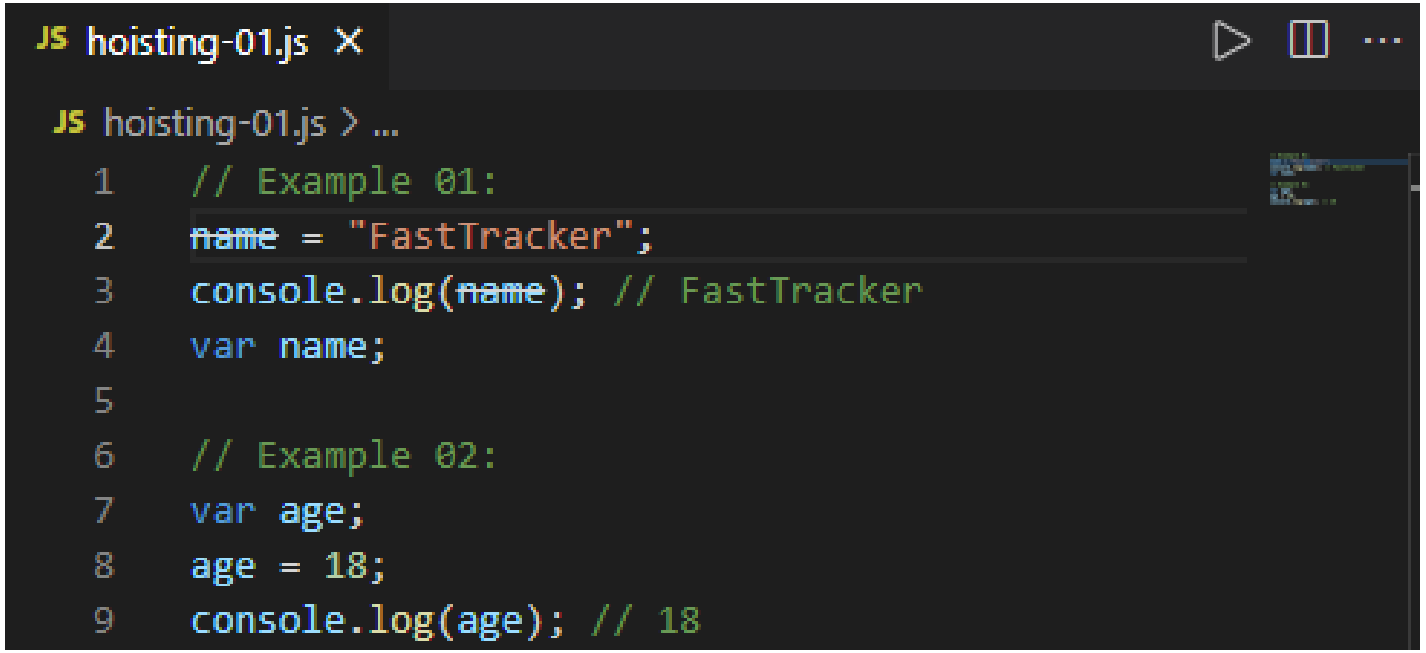
➢ **Hoisting** trong JavaScript về cơ bản là **chỉ việc trước khi bất kỳ đoạn code nào được thực thi** thì **tất cả những khai báo biến, định nghĩa hàm sẽ được "kéo" lên – "cẩu" lên** (di chuyển) **lên trên đầu của scope hiện tại**.



Phạm vi hàm (Function Scopes)

❑ 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**.

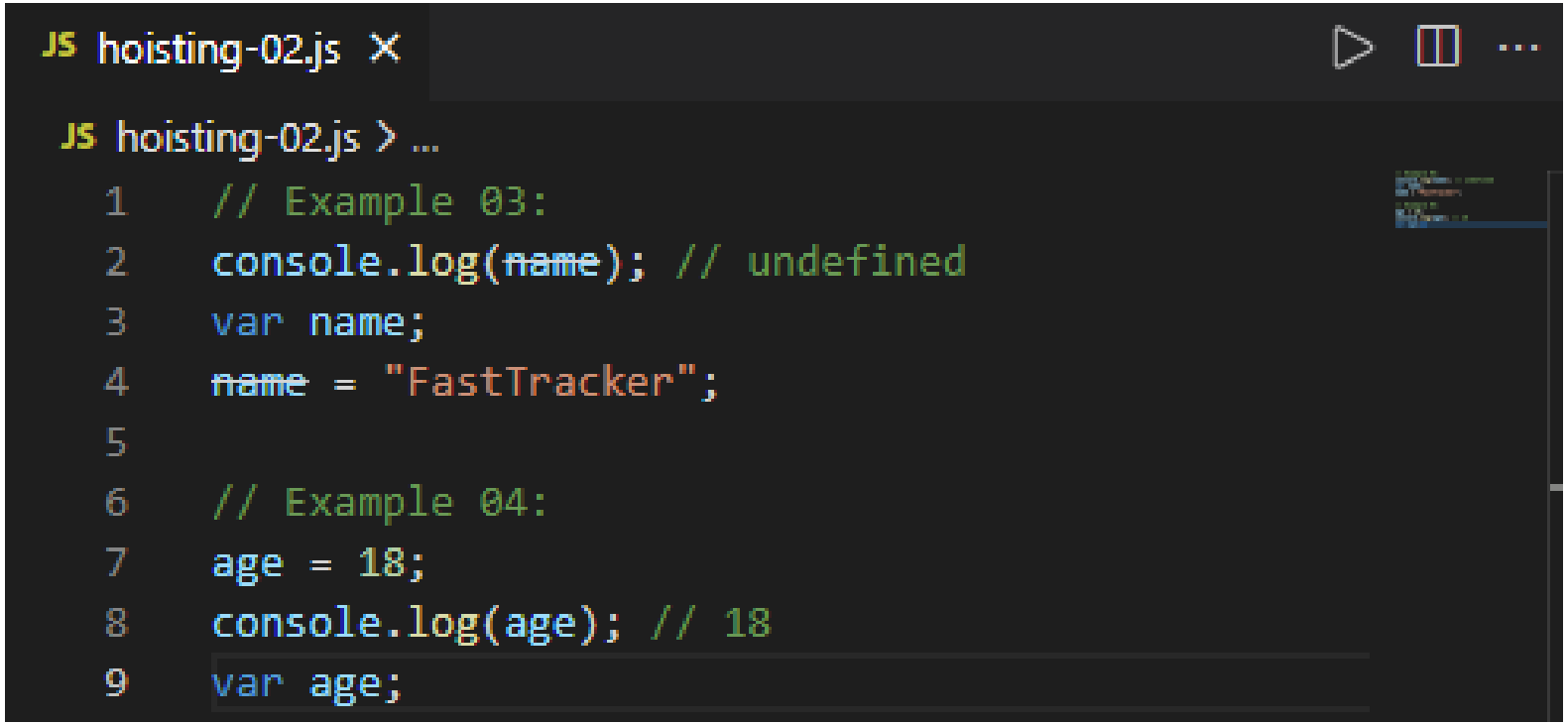


➔ Cả **2 kết quả này** cho thấy chúng **tương đương nhau**, **biến name** và **biến age đã được khai báo**.

Phạm vi hàm (Function Scopes)

❑ **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 (Function Scopes)

- **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

```
1  // Example 05:
2  function say() {
3      console.log(name);
4      var name;
5      name = 'FastTrackers';
6  }
7
8  say(); // undefined
```

hoisting-04.js

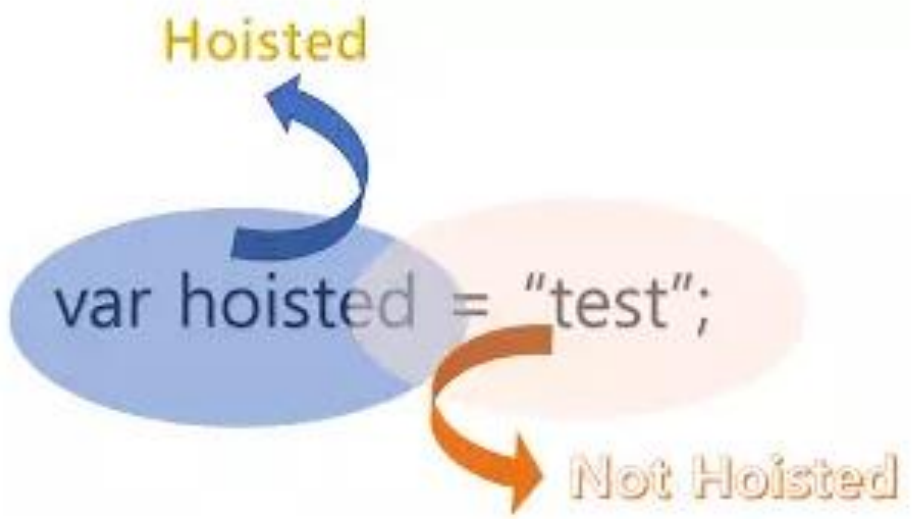
```
1  // Example 06:
2  function say() {
3      var name;
4      name = 'FastTrackers';
5      console.log(name);
6  }
7
8  say(); // FastTrackers
```

Phạm vi hàm (Function Scopes)

❑ **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**

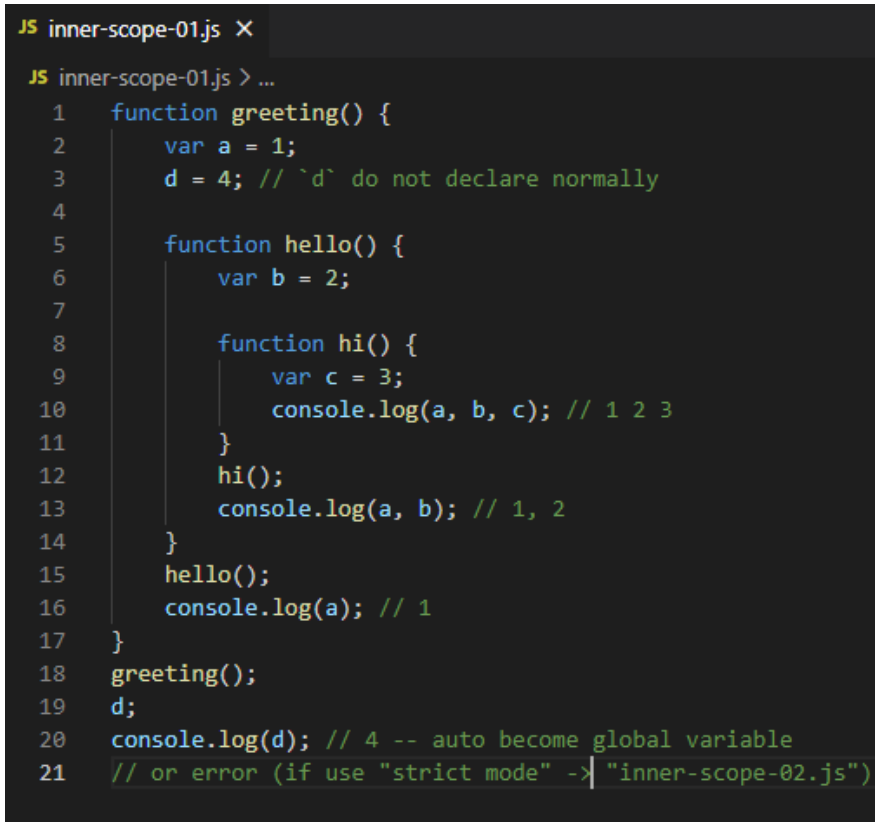


Phạm vi hàm (Function Scopes)

❑ **Phạm vi lồng nhau (Inner Scope):**

➢ Khi bạn khai báo 1 biến, nó có hiệu lực ở toàn bộ trong phạm vi đó, kể cả phạm vi con.

- **c** không có trong **hello()**, bởi vì nó **chỉ được khai báo bên trong scope hi()**
- Tương tự, **b không có trong greeting()**
- Nếu **cố tình truy cập giá trị biến trong một scope không chứa nó**, sẽ **gặp lỗi ReferenceError**.
- Nếu **cố tình lập 1 biến chưa được khởi tạo d**, bạn sẽ **vô tình tạo 1 biến ở tầng cao nhất (toàn cục)** hoặc **lỗi** (tùy "strict mode")



Phạm vi hàm (Function Scopes)

❑ **Phạm vi lồng nhau (Inner Scope):**

➢ Khi bạn khai báo 1 biến, nó có hiệu lực ở toàn bộ trong phạm vi đó, kể cả phạm vi con.

▪ Nếu **cố tình lập 1 biến chưa được khởi tạo d**, bạn sẽ **vô tình tạo 1 biến ở tầng cao nhất** (**toàn cục**) hoặc **lỗi** (tùy "strict mode")

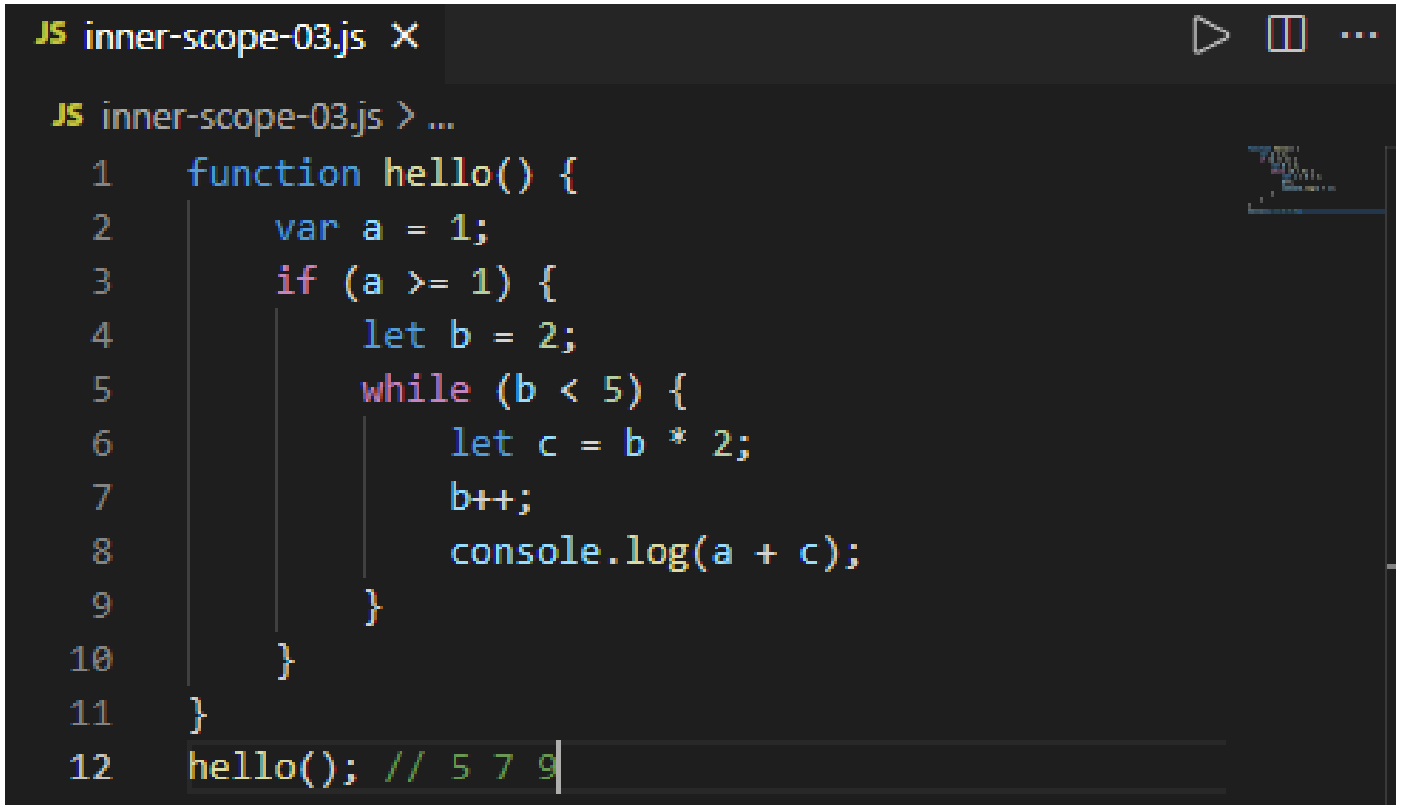
➔ **Không nên khai báo biến** một cách **ngầm định** như vậy (sẽ **dễ gây ra lỗi không mong đợi**)

inner-scope-02.js

```
1  "use strict";
2
3  function greeting() {
4      var a = 1;
5      d = 4; // `d` do not declare normally
6
7      function hello() {
8          var b = 2;
9
10         function hi() {
11             var c = 3;
12             console.log(a, b, c); // 1 2 3
13         }
14         hi();
15         console.log(a, b); // 1, 2
16     }
17     hello();
18     console.log(a); // 1
19 }
20 greeting();
21 d;
22 console.log(d); // 4 -- auto become global variable
23 // or error (if use "strict mode" -> "inner-scope-02.js")
```

Phạm vi hàm (Function Scopes)

- **Phạm vi lồng nhau (Inner Scope):**
  - **Nên khai báo biến** một cách **rõ ràng với từ khóa let** (ES6)
  - **let** cho phép **khai báo biến thuộc** về **các khối** (block) **riêng biệt**.



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**:

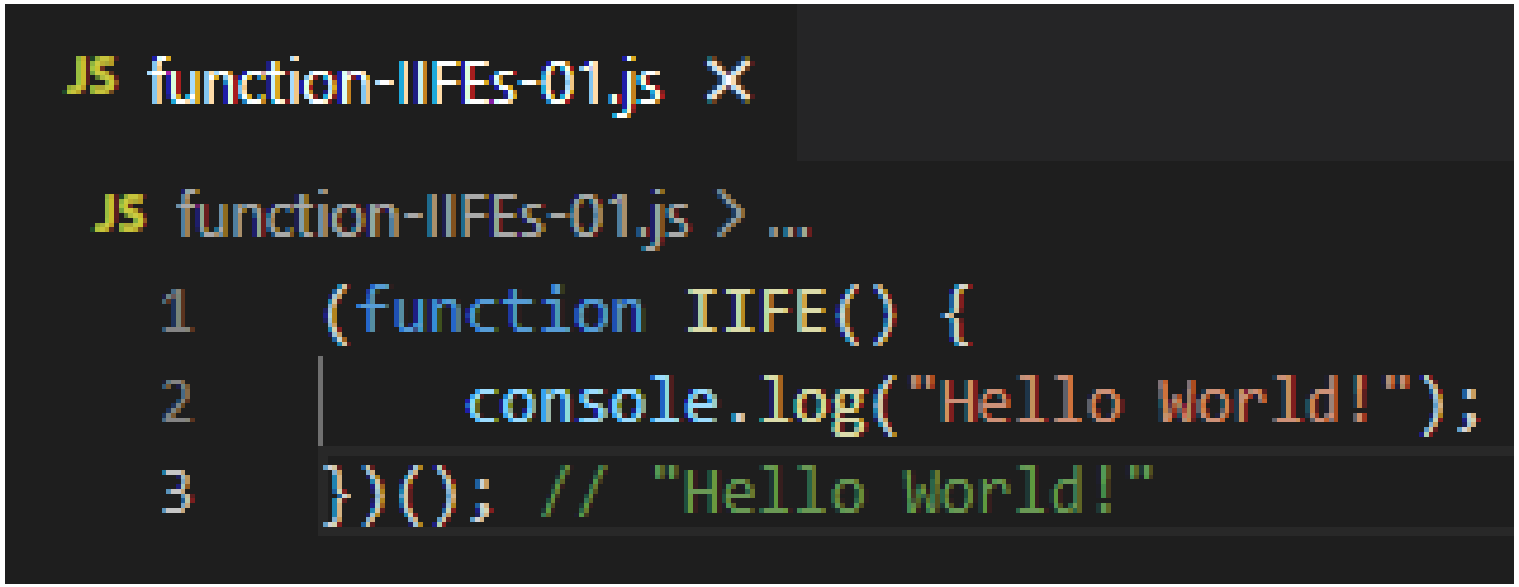
JS function-IIFEs-01.js

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

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

Hàm biểu thức IIEFs

❑ **Ví dụ**:

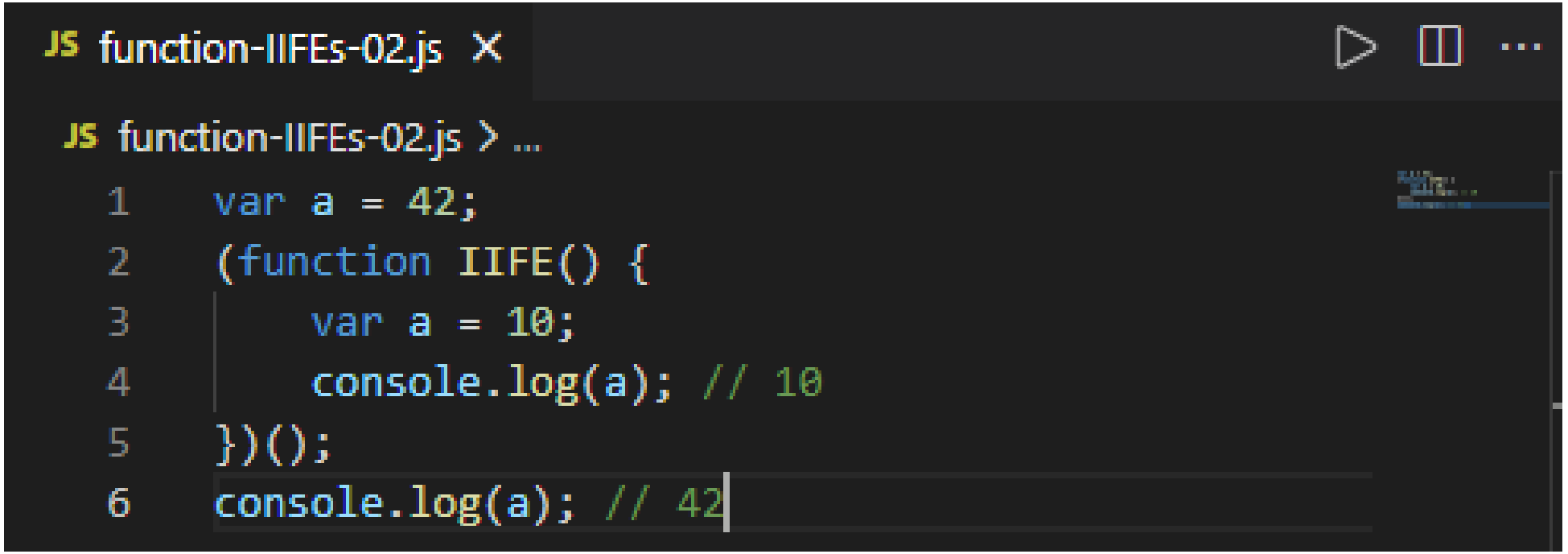


❑ Dấu **()** cuối cùng của biểu thức **})();** là chính xác **cái gì sẽ thực thi tức thì trước nó**.

❑ **()** 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**:



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**:

JS function-IIFEs-03.js

```
var x = (function IIFE() {
    return 42;
})();
console.log(x); // 42
```

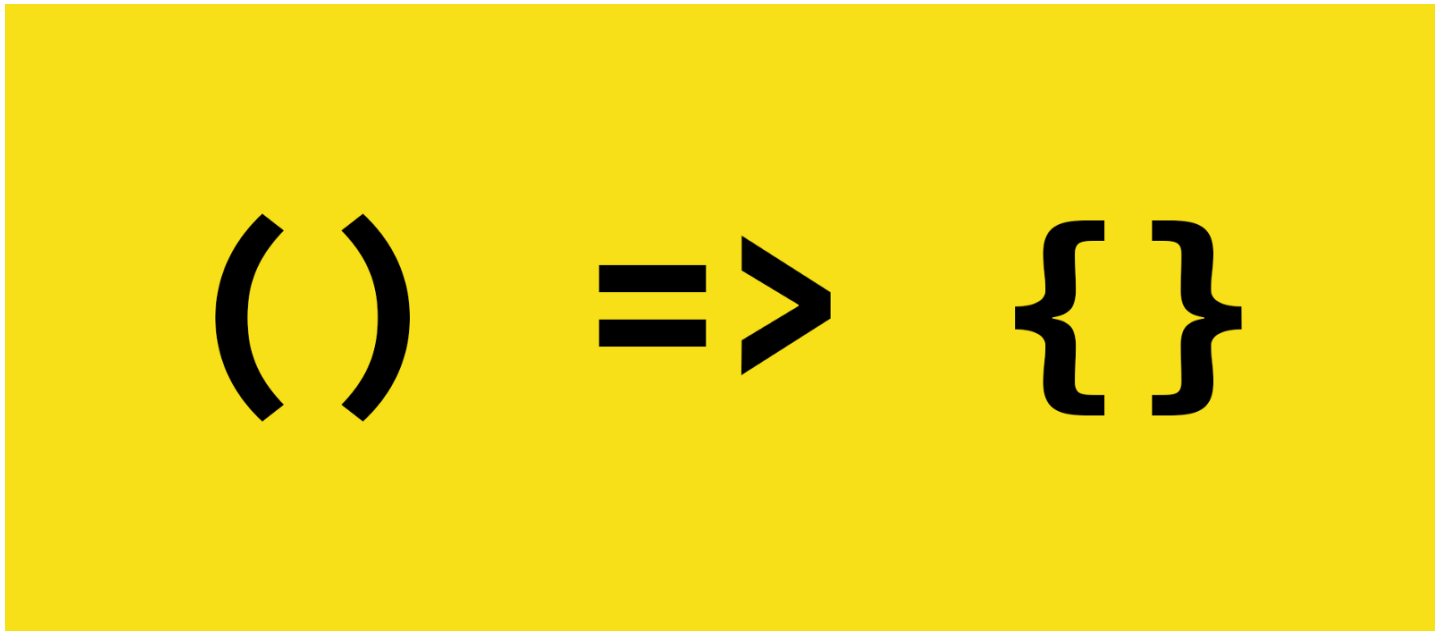
- Giá trị **42** được **return** từ **IIFE** – thực thi function được đặt tên theo **x**.

Hàm mũi tên (arrow function)

❑ **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)

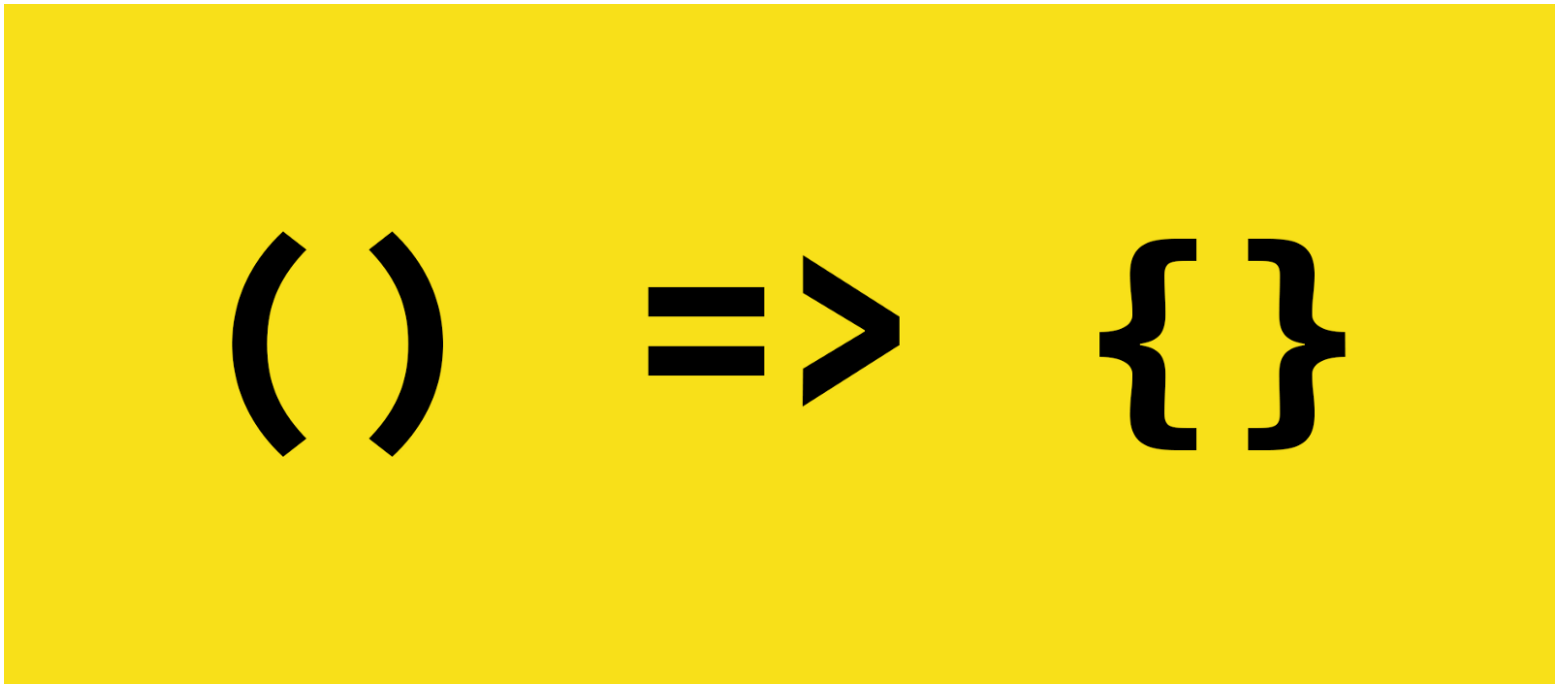


Hàm mũi tên (arrow function)

❑ **Hàm mũi tên** (arrow function):

➢ 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 (arrow function)

- **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**.

JS arrow-function-01.js

JS arrow-function-01.js > ...

```
1   // (param1, param2, paramN) => expression
2
3   // ES5
4   var multiply = function(x, y) {
5       return x * y;
6   };
7   console.log(multiply(1, 2)); // 2
8
9   // ES6
10  var multiply = (x, y) => { return x * y };
11  console.log(multiply(1, 2)); //2
12
13  // Arrow Function
14  var multiply = (x, y) => x * y;
15  console.log(multiply(1, 2)); //2
```

Hàm mũi tên (arrow function)

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

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"]
```

Hàm mũi tên (arrow function)

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

Tổng kết:

- ❑ Lệnh break & continue
- ❑ Hàm (Function), cách dùng hàm
- ❑ Viết chương trình sử dụng hàm
- ❑ Phạm vi hàm (Function Scopes)
- ❑ Hàm biểu thức IIFEs
- ❑ Hàm mũi tên (Arrow Function)