VIETNAM AUSTRALIA Vocational College

LẬP TRÌNH JAVASCRIPT CƠ BẢN

MẢNG (ARRAY), ĐỐI TƯỢNG (OBJECT) VÀ JSON

Nội dung:

1. Giá trị (Value) & kiểu (Type)
2. Đối tượng (Object)
3. Từ khóa this
4. Mảng (Array)
5. Kiểu dữ liệu JSON
6. Tham trị và tham biến

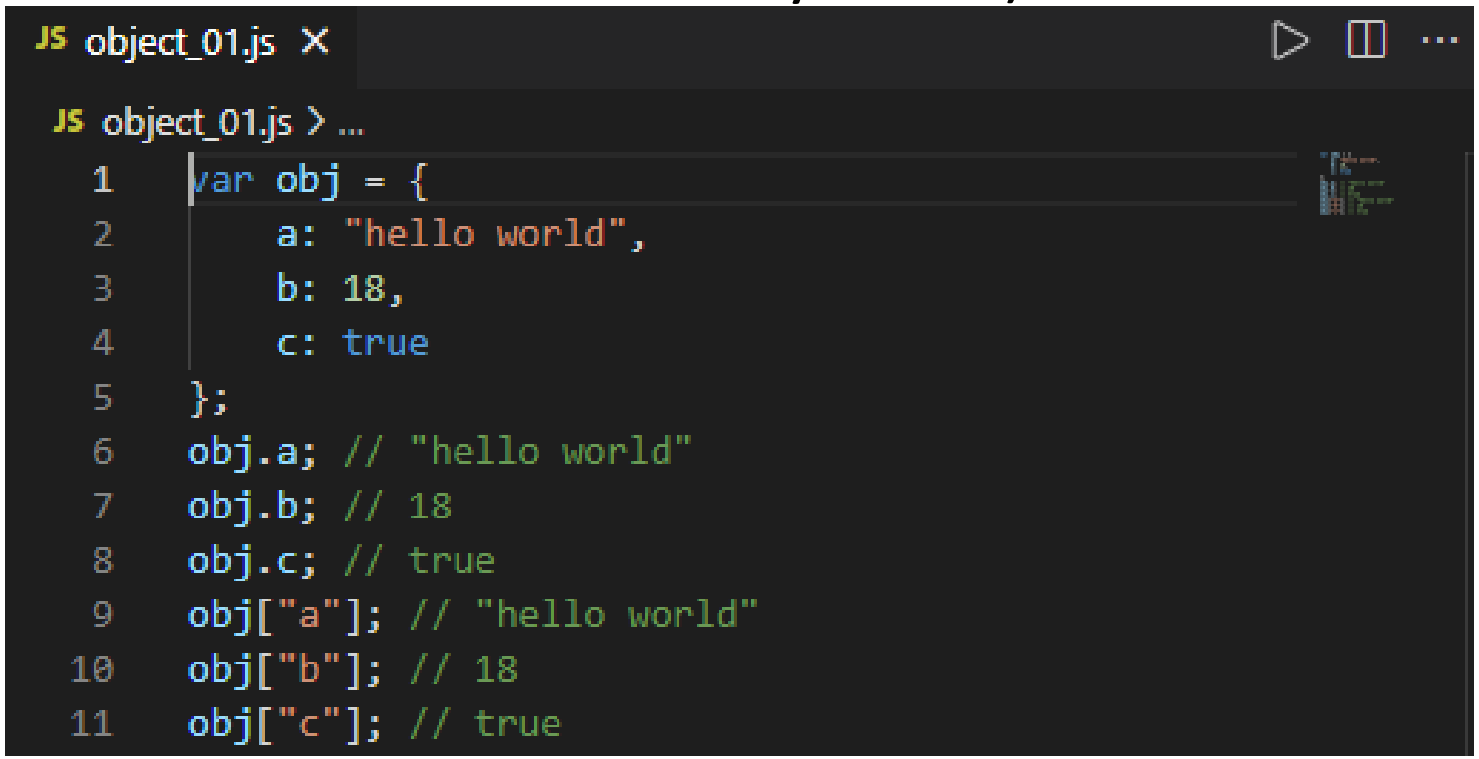
Giá trị (Value) & kiểu (Type)

- Như đã tìm hiểu về giá trị và kiểu ở bài 2, JavaScript có **kiểu của giá trị**, **không có** **kiểu của biến**.
- Các **kiểu** **có sẵn** là:
  - string
  - number
  - boolean
  - null và undefined
  - object
  - symbol (mới có trong ES6)
    - chi tiết sau khi học object

Giá trị (Value) & kiểu (Type)

- **JavaScript** có một **biểu thức typeof** có thể **kiểm tra giá trị** và **cho** bạn **biết kiểu** của **biểu thức đó** là gì.

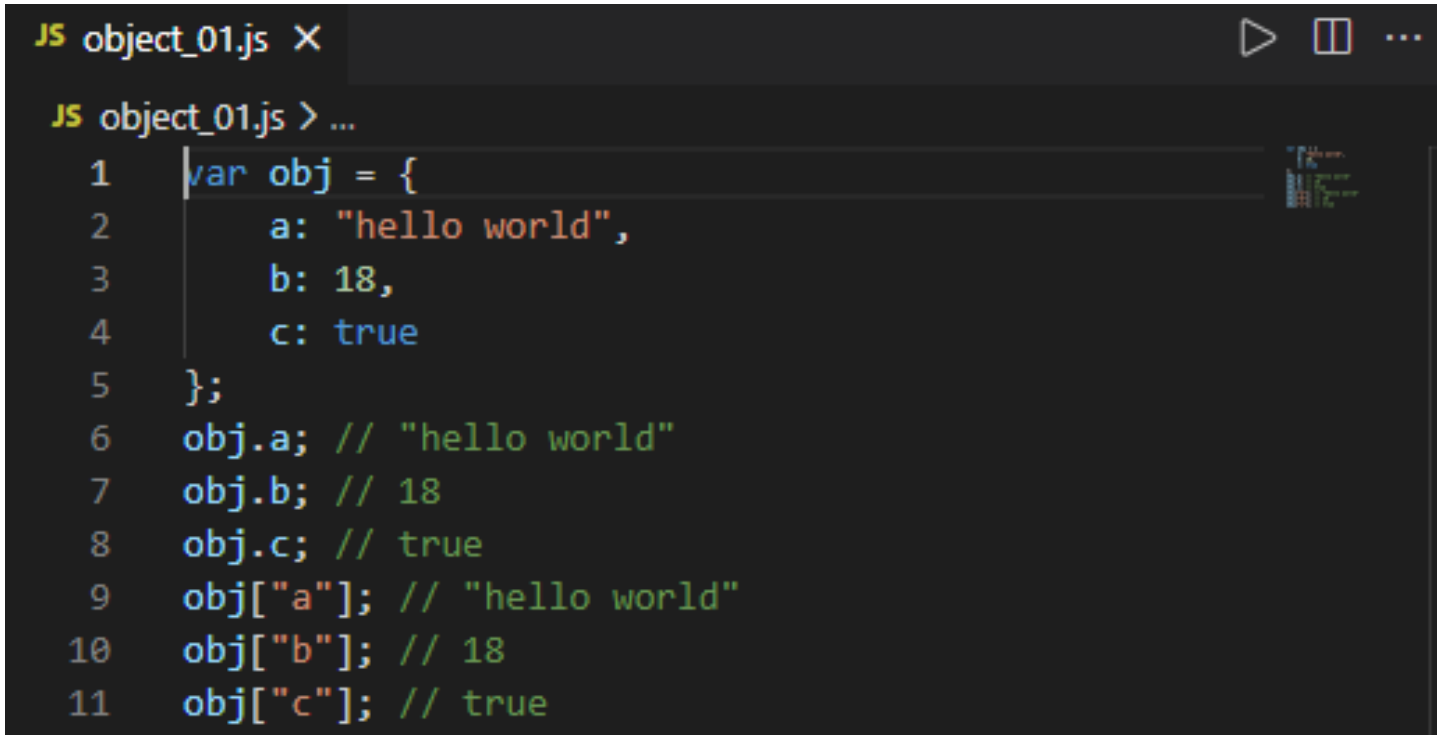
  - **Giá trị trả lại từ biểu thức typeof** luôn là **1 trong 6 kiểu** ở **dạng giá trị string** (ES6 là 7 kiểu – thêm kiểm "symbol")



Giá trị (Value) & kiểu (Type)

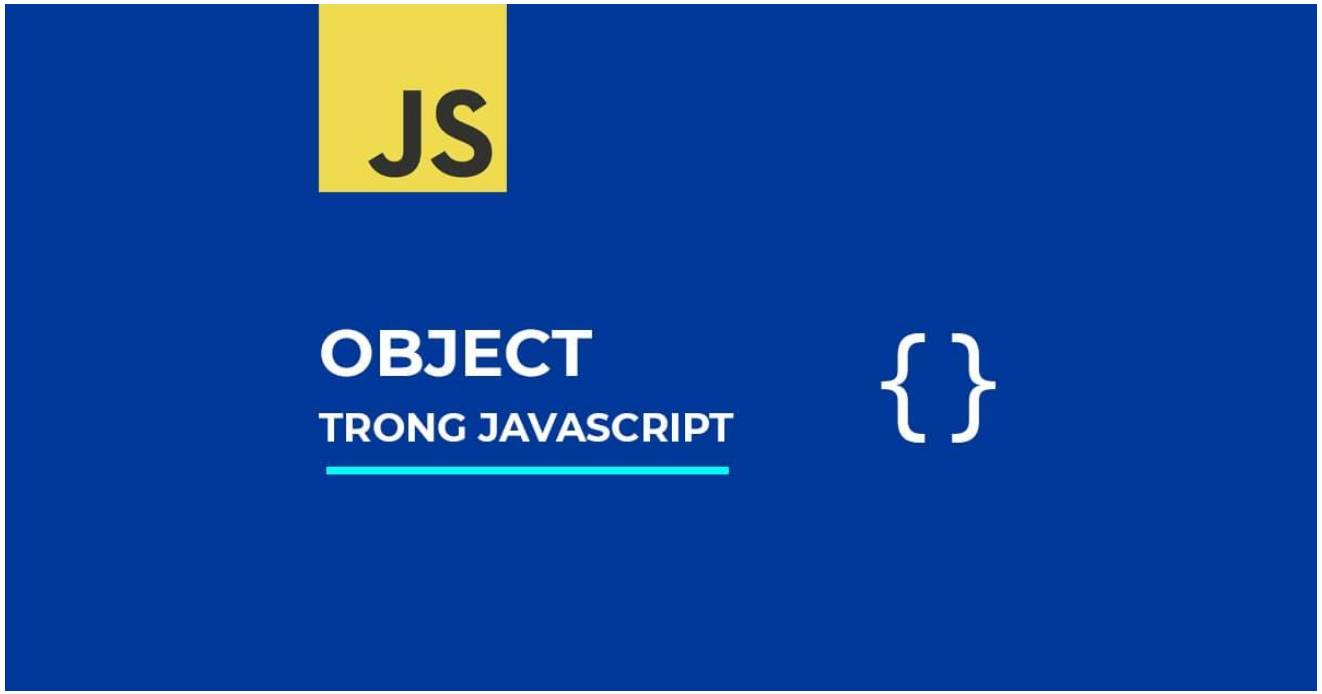
➢ **Biến box giữ mọi kiểu của giá trị**, **typeof box** sẽ trả về **kiểu** của **giá trị hiện tại trong biến box**, chứ không phải kiểu của biến box.

➢Lưu ý: **typeof null** là **trường hợp đặc biệt**, nó trả về sai kiểu thành **“object”**, khi mong muốn trả về **“null”**.



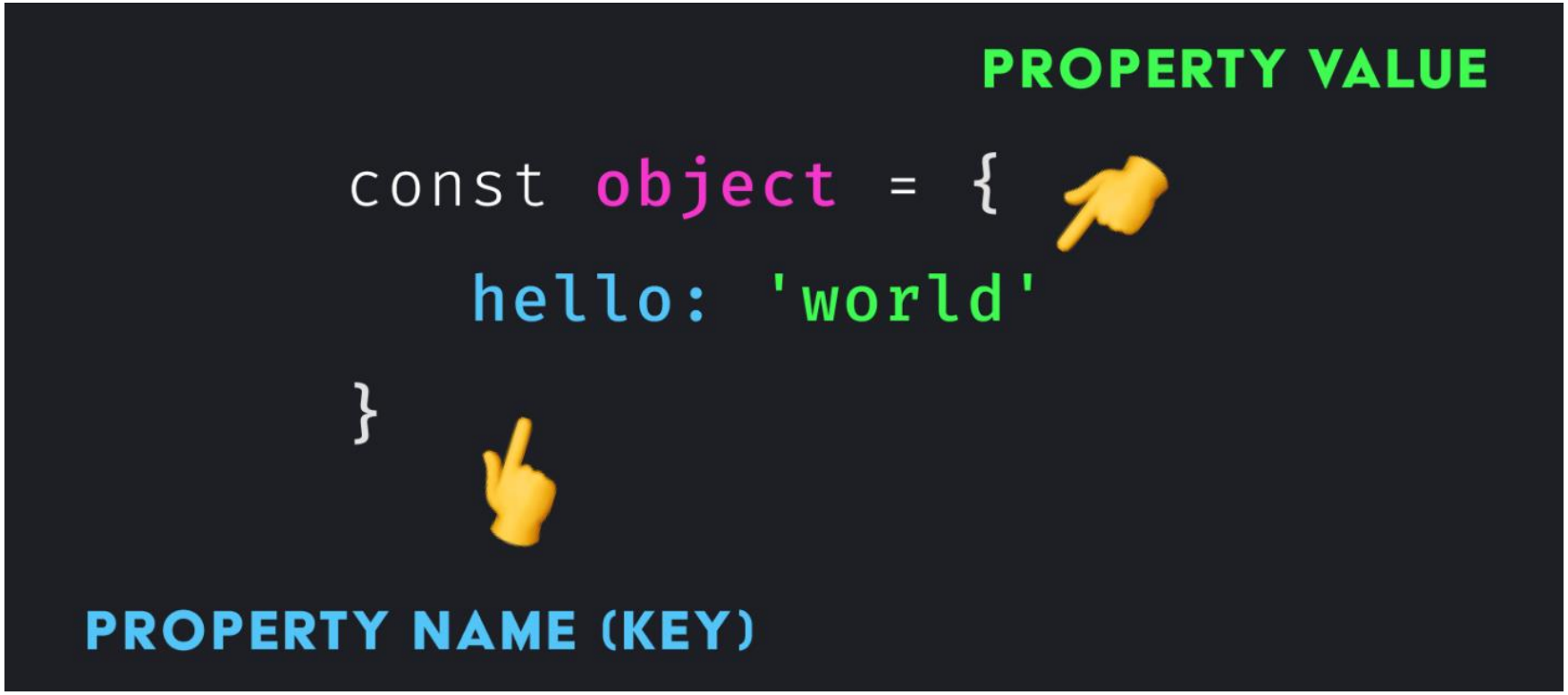
Đối tượng (Object)

- Kiểu **object** đề cập đến **một giá trị phức hợp** mà bạn **có thể lập các thuộc tính**.
- Mỗi **thuộc tính** đều **có thể có giá trị của riêng chúng** với **bất kỳ kiểu nào**.
- Kiểu **object** là **một trong những kiểu hữu dụng nhất** trong JavaScript.



Đối tượng (Object)

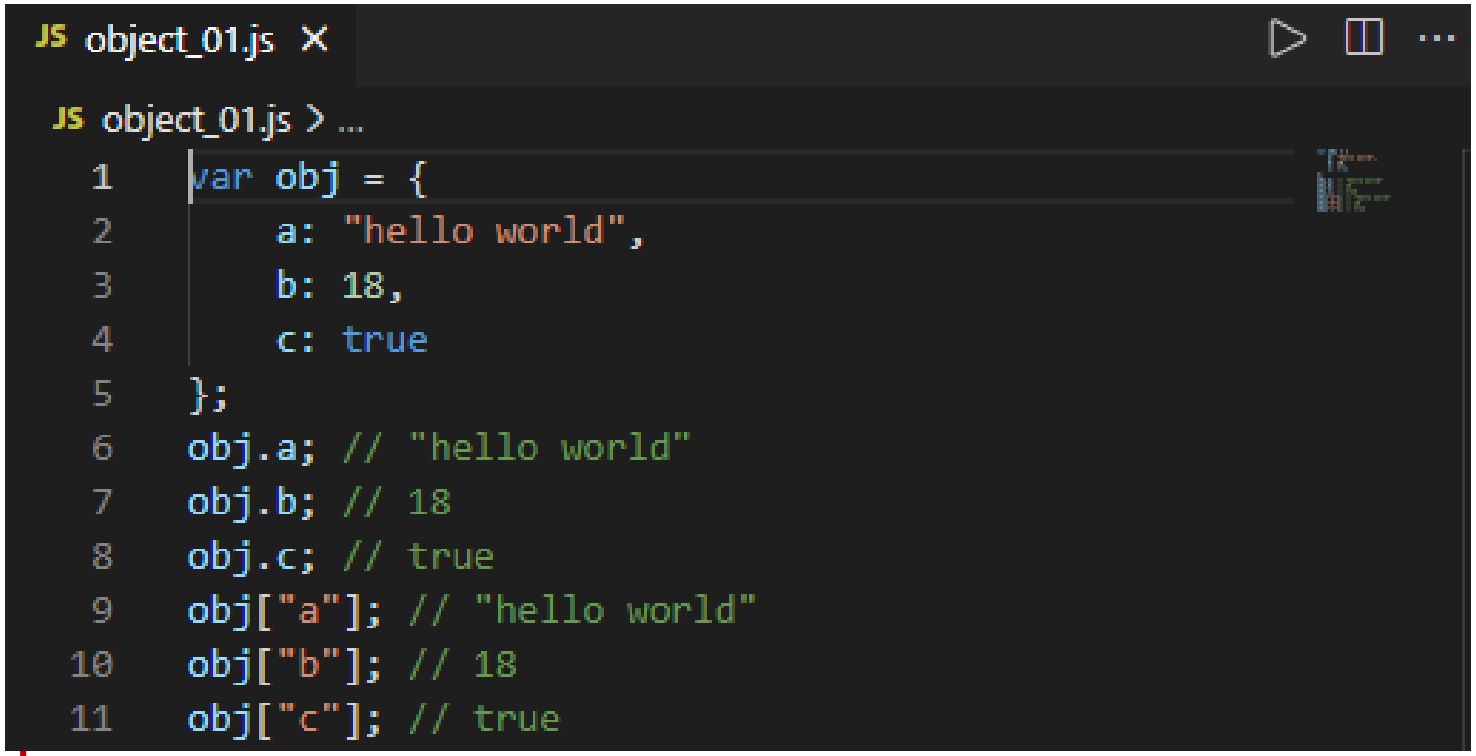
- **Object** chứa nhiều giá trị **gồm các thuộc tính** và **phương thức**.
- **Object** được **tạo bằng cặp ngoặc nhọn**.
- **Thuộc tính** trong **object gồm key** và **value**.



Đối tượng (Object)

❑ **Cú pháp**:

```
var <object-name> = {
    key1: value1,
    key2: value2,
    ...
    keyN: valueN
};
```



❑ **Trong đó**:

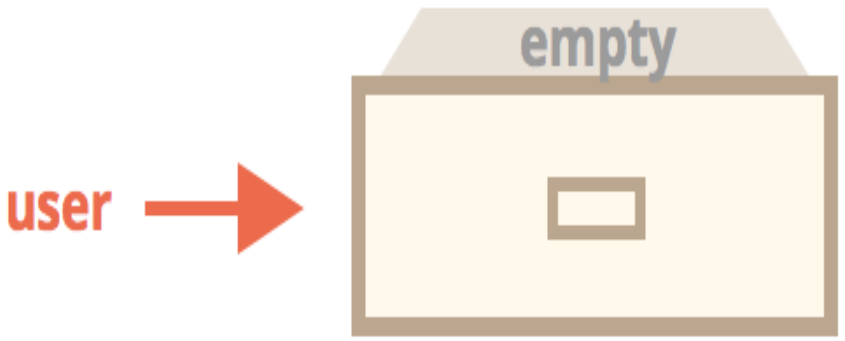
➢ **key**: property/method

➢ **method**: function/function expression vai trò property trong **object**.

Đối tượng (Object)

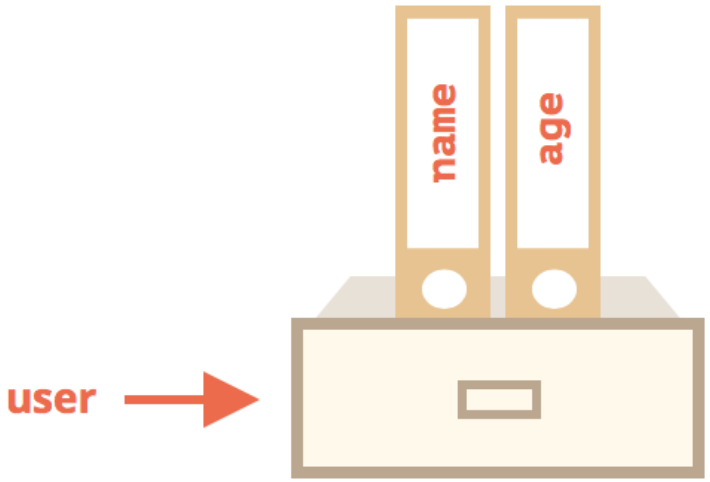
- Tạo 1 object rỗng:

```
var emptyObject = {
    // nothing here
    // ...
};
```



- Tạo 1 object user có 2 thuộc tính:

```
let user = {
    name: "Teo",
    age: 18
};
```



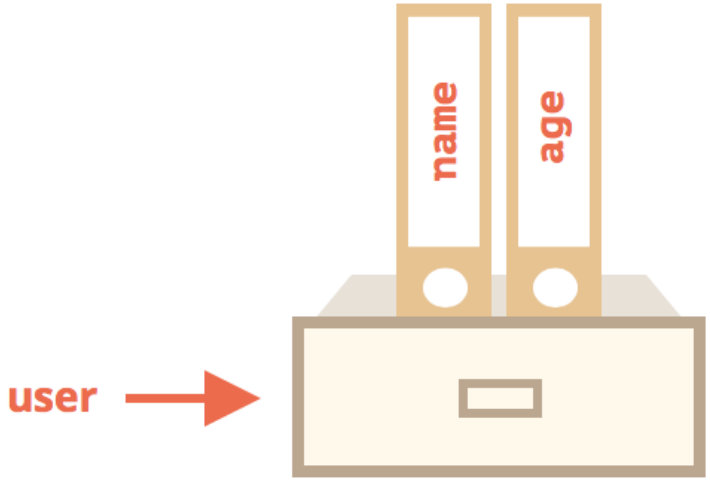
Đối tượng (Object)

- **Tạo 1 object user** có **2 thuộc tính** và **1 biểu thức hàm**:

```
var user = {
    name: "Teo",
    age: 18,
    sayHello: function () {
        console.log("Hello World!");
    }
};
user.sayHello(); // Hello World!
```

properties

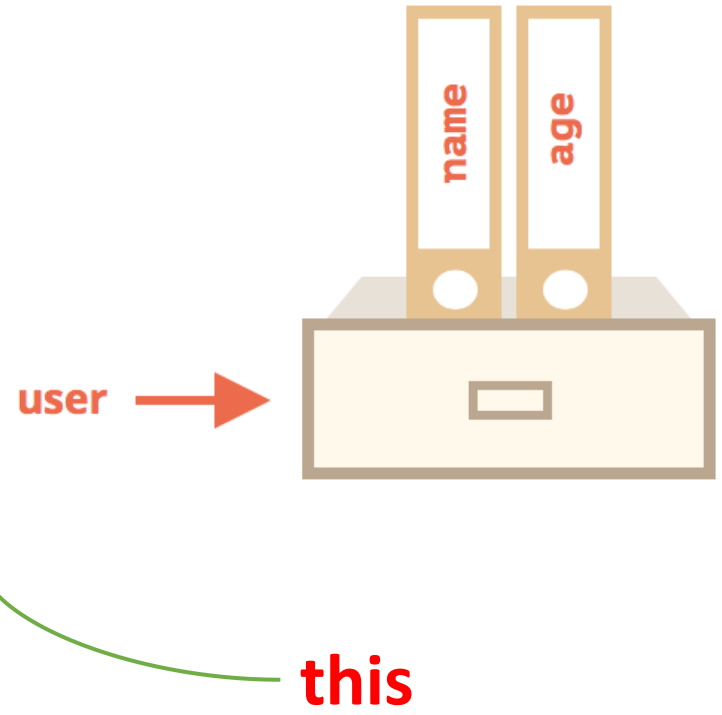
method (function expression)



Đối tượng (Object)

- Khi **1 method của object muốn truy xuất các properties của chính object đó**, ta nên **sử dụng từ khóa this**:

```
var user = {
    name: "Teo",
    age: 18,
    sayName: function () {
        console.log(this.name);
    }
};
user.sayName(); // Teo
```



Đối tượng (Object)

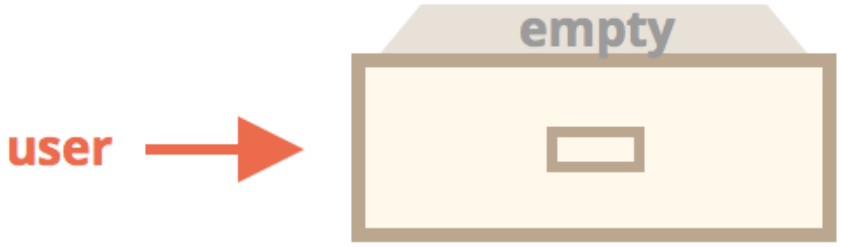
❑ Tạo 1 object với từ khóa new:

- ➢ Để tạo 1 object, ta còn có cách sử dụng từ khóa new như sau:

  var <object-name> = new Object();

- ➢ Tạo 1 object rỗng:

  var emptyObject = new Object();




```
JS new_01.js  ×

JS new_01.js > ...
1    var emptyObject = new Object();
2    console.log(emptyObject); // {}
```

Đối tượng (Object)

- Quan sát **giá trị** của **obj** một cách **trực quan** như sau:

obj

a: **"hello world"**	**b**: **18**	**c**: **true**

- **Thuộc tính** có thể **được truy cập bằng 2 cách**:
  - Dấu **chấm** (.) -> Ví dụ: **obj.a**
  - Dấu **ngoặc** ([]) -> Ví dụ: **obj["a"]**
- Dấu **chấm** **ngắn hơn** và **dễ đọc hơn** nên **thường được ưa thích hơn**.
- Dấu **ngoặc** **hữu dụng** khi bạn có **một tên thuộc tính có ký tự đặc biệt** trong đó (Ví dụ: obj["hello world**!**"])

Đối tượng (Object)

- Quan sát **giá trị** của **obj** một cách **trực quan** như sau:

obj

a: “hello world”	**b**: 18	**c**: true

- Dấu **ngoặc hữu dụng** khi bạn có **một tên thuộc tính có ký tự đặc biệt** trong đó (Ví dụ: **obj["hello world!"]**)
- Dấu **ngoặc yêu cầu** phải có **một biến** hoặc **một string nguyên bản**.
- Một **string nguyên bản** thường **được bao bởi**:
  - Dấu **nháy kép** “..”
  - Dấu **nháy đơn** ‘..’

Đối tượng (Object)

- Quan sát **giá trị** của **obj** một cách **trực quan** như sau:

obj

a: “hello world”	**b**: 18	**c**: true

- Dấu **ngoặc hữu dụng** khi bạn **muốn tiếp cận** một **thuộc tính/chìa khóa (key) nhưng tên được lưu trữ ở biến khác**. Ví dụ:

object_02.js

```
1  var obj = {
2      a: "hello world",
3      b: 42
4  };
5  var b = "a";
6  obj[b]; // "hello world" ~ obj["a"]
7  obj["b"]; // 42
```

Đối tượng (Object)

❑ **Thêm thuộc tính vào object:**

➢ Để **thêm mới thuộc tính** vào **object**, ta **dùng tên object** và **chấm (.) tên thuộc tính mới** muốn thêm vào **object**.

➢ **Ví dụ:**

object_06.js

```
var user = {
    name: "Teo",
    age: 18
};
user.isAdmin = true;
console.log(user); //{ name: 'Teo', age: 18, isAdmin: true }
```

Đối tượng (Object)

❑ **Xóa thuộc tính của object**:

➢ Để **xóa thuộc tính** của **object**, ta **dùng từ khóa delete, khoảng trắng và chấm (.) tên thuộc tính của object** muốn xóa.

➢ **Ví dụ**:

object_07.js

```
var user = {
    name: "Teo",
    age: 18
};
user.isAdmin = true;
console.log(user); //{ name: 'Teo', age: 18, isAdmin: true }

delete user.age;
console.log(user); //{ name: 'Teo', isAdmin: true }
```

Đối tượng (Object)

❑ **Lặp thuộc tính của object:**

➢ Để **lặp qua từng thuộc tính (key)** của một **object**, ta **dùng cấu trúc** sau: **key in object**

➢ **Ví dụ:**

object_08.js

```
var user = {
    name: "Ty",
    age: 20
};
user.isAdmin = true;

for (let key in user) {
    console.log(key);
} // name age isAdmin
```

Đối tượng (Object)

- **keys, values, entries của object**:
  - **Một số hỗ trợ** của **đối tượng** (object) **phổ biến** như sau:
    - **Object.keys(obj)**: trả về **mảng các keys**
    - **Object.values(obj)**: trả về **mảng các values**
    - **Object.entries(obj)**: trả về **mảng các cặp [key, value]**

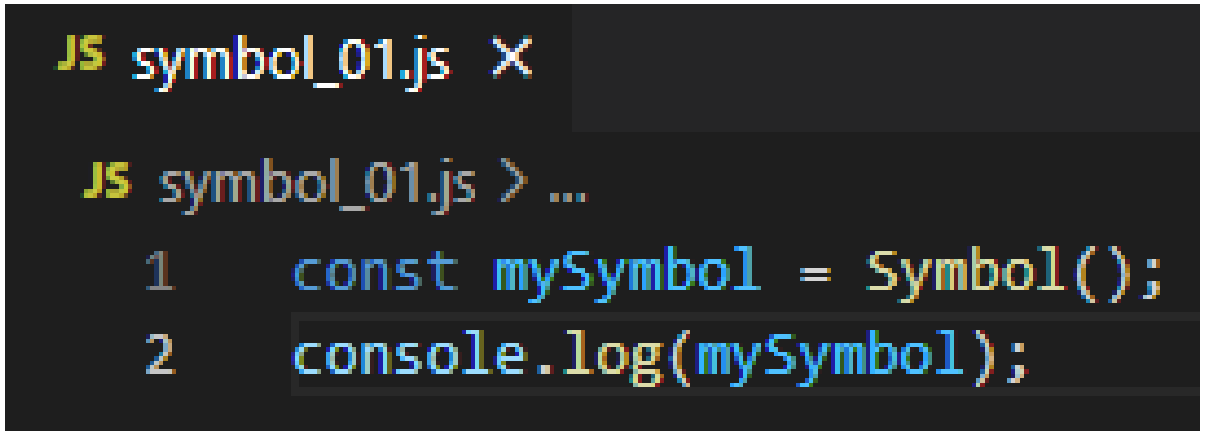
object_09.js

```
var user = {
    name: "Teo",
    age: 20
};

console.log(Object.keys(user)); // ['name', 'age']
console.log(Object.values(user)); // ['Teo', 20]
console.log(Object.entries(user)); // [['name', 'Teo'], ['age', 20]]
```

Đối tượng (Object)

- Sau khi học về **Object**, ta quay lại 1 **kiểu dữ liệu symbol** đã nhắc đến ở đầu bài học.
- **Symbol** là **một kiểu dữ liệu nguyên thủy** của **JavaScript**.
- **Symbol** là **một kiểu đặc biệt xuất hiện lần đầu** tiên **trong ES6**.
- **Symbol có giá trị được giữ kín** và **chỉ được sử dụng nội bộ**.
- **Symbol** được **tạo ra bằng cách gọi hàm global Symbol()**. Ví dụ:



Đối tượng (Object)

- Mỗi **khi gọi hàm Symbol()**, chúng ta sẽ **nhận** được **một giá trị mới** và giá trị này là **duy nhất** (unique).

symbol_02.js

```
1  var check = Symbol() === Symbol();
2  console.log(check); //false
```

- Ta **có thể truyền tham số cho Symbol()** và được **sử dụng làm mô tả cho chính Symbol đó**.

symbol_03.js

```
1  console.log(Symbol()); // Symbol()
2  console.log(Symbol('Name')); //Symbol(Name)
```

Đối tượng (Object)

❑ **Symbol** thường được sử dụng **để xác định các thuộc tính đối tượng** vì:

➢ **Tránh** được **xung đột tên** giữa **các thuộc tính** của **các đối tượng** (vì không có **Symbol** nào giống nhau cả)

JS symbol_04.js ×

JS symbol_04.js > ...

```
1   const NAME = Symbol();
2   const person = {
3       [NAME]: 'Teo'
4   };
5   person[NAME]; //'Teo'
6   console.log(person[NAME]); //'Teo'
7
8   const RUN = Symbol();
9   person[RUN] = () => 'Person is running';
10  console.log(person[RUN]()); //'Person is running'
```

Mảng (Array)

- **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:

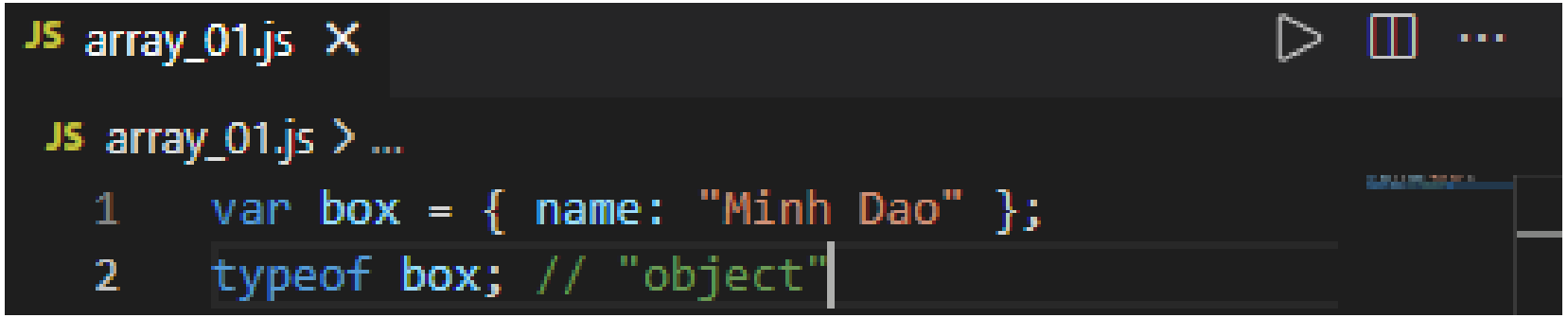
arr

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 (Array)

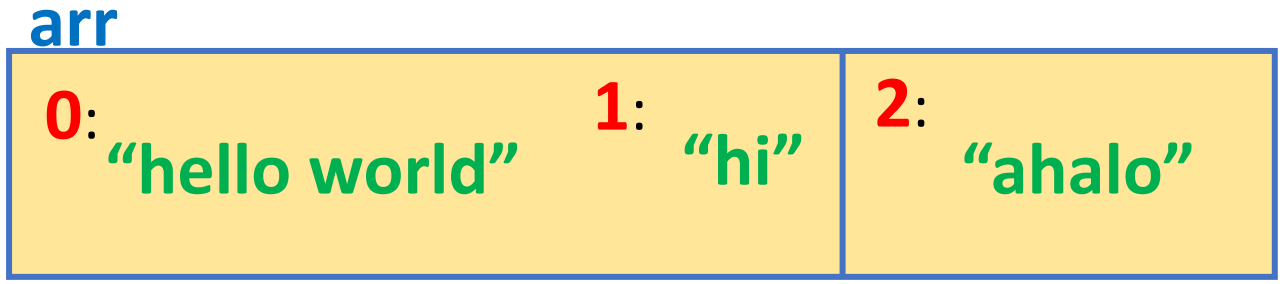
- **Mảng** là **một object đặc biệt** (**như typeof** đã **ngụ ý** ở ví dụ dưới đây)



- **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 (Array)

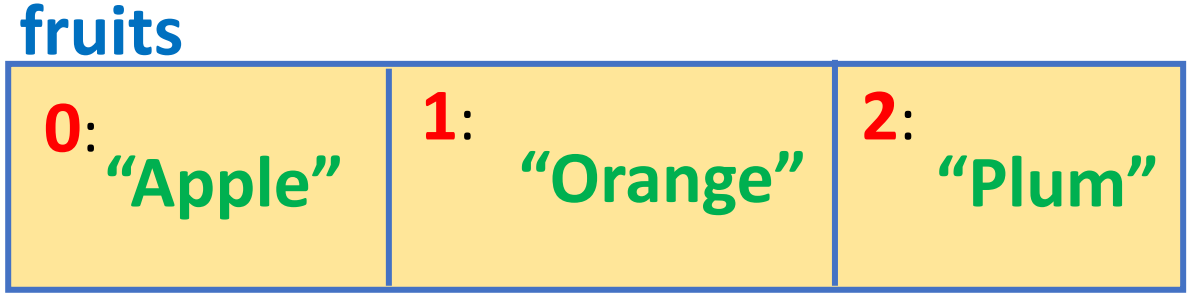
- Về bản chất, **mảng** là **một object đặc biệt**. Nhưng chúng ta **thường sử dụng mảng với một số đặc tính riêng biệt** để hỗ trợ các vấn đề lập trình cụ thể liên quan với mảng.
- Một số **đặc tính mảng** cần lưu ý như sau:
  - Mảng là **một dãy các phần tử** có **cùng cấu trúc** và **lưu trữ liên 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 (Array)

- **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 (Array)

- 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 (Array)

- 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 (Array)

- 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 (Array)

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

array_06.html | array_06.js

```
// Input array:
var arr = [];
var n;

function input() {
    n = prompt("Enter number n: ");
    for (var i = 0; i < n; i++) {
        arr[i] = prompt("Enter a number arr[" + i + "]: ");
    }
}

// Output array:
function output() {
    for (var i = 0; i < n; i++) {
        document.write("a[" + i + "]: " + arr[i] + "<br>");
    }
}

// Call function:
input(arr, n);
output(arr, n);
```

Mảng (Array)

- 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"]
```

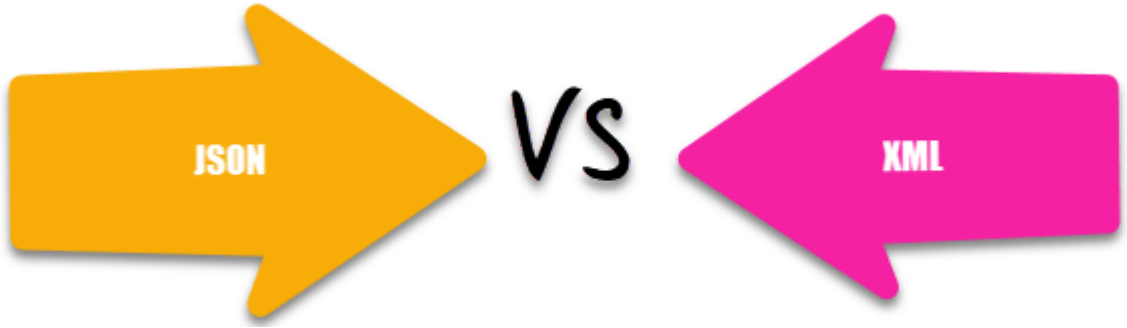
Kiểu dữ liệu JSON

❑ **Định nghĩa**:

➢ **JSON** (JavaScript Object Notation) là **định dạng giao đổi dữ liệu** (phổ biến) **cực kỳ nhẹ**, được dùng **để giao đổi dữ liệu giữa máy chủ và máy khách**.

➢ **Giống** như **XML**, **JSON** là **một dạng định dạng dựa trên văn bản** dễ viết và dễ hiểu cho cả người và máy tính.

➢ Nhưng **không giống XML**, cấu trúc **JSON chiếm ít băng thông hơn các phiên bản XML**.



Kiểu dữ liệu JSON

- **JSON** thường được dùng trong các ngôn ngữ lập trình hiện đại (trong đó có JavaScript).
  - **JavaScript** cung cấp **phương thức JSON.stringify()** để **chuyển đổi giá trị JavaScript thành một chuỗi JSON** như sau:

Kiểu dữ liệu JSON

- **JSON** thường được dùng trong các ngôn ngữ lập trình hiện đại (trong đó có JavaScript).
  - **JavaScript** cung cấp **phương thức JSON.parse ()** để **chuyển đổi chuỗi JSON thành object trong JavaScript** như sau:

Tham trị và tham chiếu

❑ **Tham trị với primitive data type**:

➢ Có **6 kiểu dữ liệu nguyên thủy** (primitive data type): **undefined**, **boolean**, **number**, **string**, **bigint**, **symbol**.

➢ Khi **ta gán giá trị biến này cho biến khác** thì **giá trị của 2 biến này độc lập** và **không liên quan với nhau** nữa.

➢ **Ví dụ**:

primitive-types-01.js

```
1   let a = 1;
2   let b = a;
3   b = 2;
4   console.log(a); // 1
5   console.log(b); // 2
```

Tham trị và tham chiếu

- **Tham trị với primitive data type**:
  - **Ví dụ**:

primitive-types-01.js

```
1   let a = 1;
2   let b = a;
3   b = 2;
4   console.log(a); // 1
5   console.log(b); // 2
```

  - Dù gán **b = a**, nhưng khi **b** thay đổi thì **a** vẫn không thay đổi.
  - Khi **giá trị thuộc kiểu dữ liệu nguyên thủy**, **biến sẽ chứa giá trị của biến đó**.

Tham trị và tham chiếu

❑ **Tham chiếu với Object**:

➢ Trong **JavaScript**: **object, array, function** đều **được coi là object**.

➢ Khi **gán 1 biến thuộc kiểu object** thì **biến đó** chỉ **lưu địa chỉ của giá trị đó trên vùng nhớ, không lưu giá trị được gán**.

➢ **Ví dụ**:

reference_types.js

```
let cars1 = ['BMW', 'Mercedes'];
let cars2 = cars1;
cars2 = ['Toyota', 'Hyundai'];
console.log(cars1); // ['BMW', 'Mercedes']
console.log(cars2); // ['Toyota', 'Hyundai']
```

Tổng kết:

- ❑ Giá trị (Value) & kiểu (Type)
- ❑ Đối tượng (Object)
- ❑ Mảng (Array)
- ❑ Các phương thức kiểu dựng sẵn
- ❑ Phạm vi hàm (Function Scopes)
- ❑ Kiểu định dạng dữ liệu JSON
- ❑ Tham trị và tham chiếu