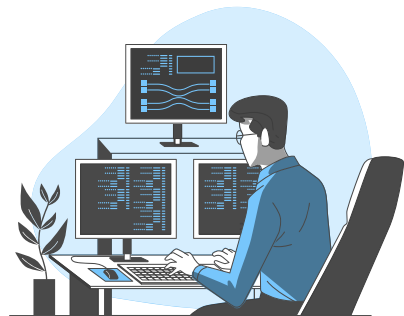


JS



Khóa học Backend

Bài 05: Javascript cơ bản (Tiết 1)



Nội dung

01

Khái niệm và giới thiệu

...

02

Variables (Biến)

...

03

Operators (Toán tử)

...

04

Data Types (Kiểu dữ liệu)

...

05

Một số hàm built-in

...

06

Typeof

...

07

Làm việc với String

...

08

Làm việc với Number

...

09

Làm việc với Array

...

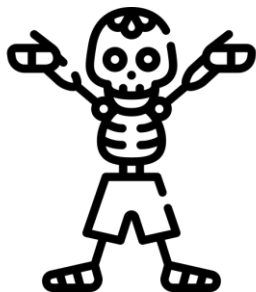


01

Khái niệm và giới thiệu

1.1. Khái niệm

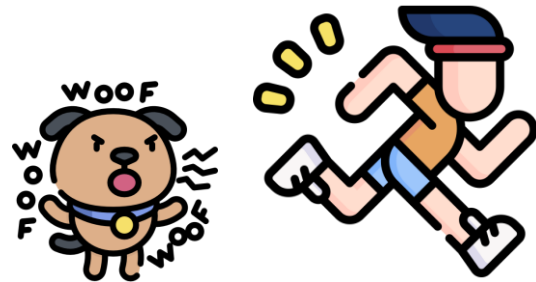
- **Javascript** (viết tắt : **JS**) là một **ngôn ngữ lập trình kịch bản** dựa vào các đối tượng có sẵn hoặc do lập trình viên tự định nghĩa.



HTML



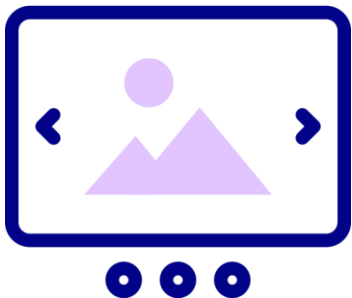
CSS



Javascript

1.2. Giới thiệu

- **Trước đây:** được sử dụng chủ yếu để nâng cao sự tương tác của người dùng với trang web.
- Ví dụ:



Slider



Popup



Form Validate

1.2. Giới thiệu

- **Ngày nay:** Sử dụng rộng rãi trong nhiều lĩnh vực.
- Ví dụ:



Web app:
Reactjs, Vuejs, Angularjs...



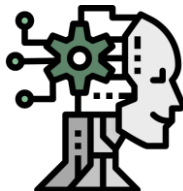
Mobile app:
React Native,...



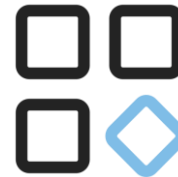
Server app:
Nodejs, Expressjs



Graphic:
two.js (2D), three.js (3D)...



AI:
brain.js...

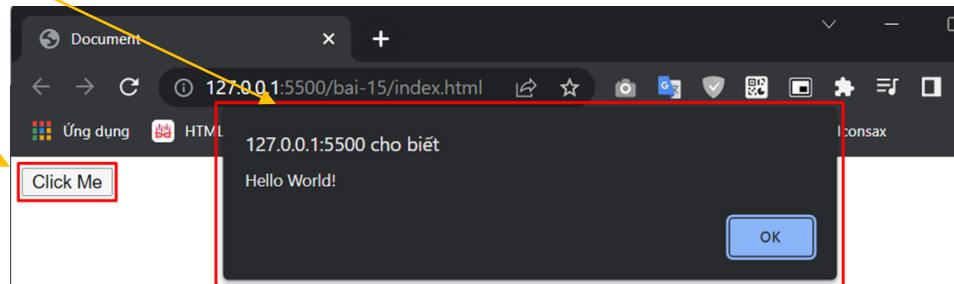


Nhiều lĩnh vực khác

1.3. Viết chương trình Hello World!

- Ví dụ: Viết chương trình Hello World!

```
<button onclick="alert('Hello World!')">  
  Click Me  
</button>
```



1.4. Các cách sử dụng javascript

- Cách 1: **Inline** - viết trực tiếp trong thẻ HTML

```
<button onclick="alert('Hello World!')">  
  Click Me  
</button>
```


1.4. Các cách sử dụng javascript

- Cách 2: **Internal** - viết trong file HTML hiện tại

```
<script>
  var button = document.querySelector("button");
  button.onclick = function () {
    alert("Hello World!");
  }
</script>
```

1.4. Các cách sử dụng javascript

- Cách 3: **External** - viết ra một file js khác rồi nhúng vào trang HTML

```
<script src="duong_dan_file"></script>
```



02

Variables (Biến)



2.1. Khái niệm

- **Biến** (variable) là nơi để bạn có thể **lưu trữ thông tin**.
- Cú pháp: **var tenBien = giaTri;**
- Trong đó:
 - **tenBien**: Là tên của biến các bạn muốn đặt.
 - **giaTri**: Là giá trị của biến, có thể là số, chuỗi, mảng, object.
- **Lưu ý**: Tên biến có phân biệt chữ hoa, chữ thường.
- Ví dụ:

```
var a = 15;  
var b = 20;  
var c = a + b; // Kết quả c = 35
```

2.2. Cách đặt tên cho biến

- **Quy tắc** đặt tên biến:
 - Bắt đầu phải là chữ cái hoặc ký tự '_'
 - Không được bắt đầu bằng số (0 → 9).
 - Không được bắt đầu bằng ký tự đặc biệt (ví dụ: #, %, ^, -)
- Đặt tên biến **ĐÚNG**:

```
var tenbien = 10;  
var _tenbien = 10;  
var TenBien = 10; // PascalCase  
var ten_bien = 10; // snake_case  
var tenBien = 10; // camelCase  
var tenbien123 = 10;
```

- Đặt tên biến **SAI**:

```
var 123tenbien = 10; // Sai do số đứng đầu  
var -tenbien = 10; // Sai do có ký tự -
```

03

Operators (Toán tử)

3.1. Arithmetic (Toán tử số học)

- Toán tử số học là toán tử dùng để thực hiện các phép toán số học.
- Danh sách các toán tử số học:

Toán tử	Mô tả
+	- Phép cộng. - Nếu là chuỗi thì nó sẽ thực hiện thao tác nối chuỗi.
-	- Phép trừ.
*	- Phép nhân.
**	- Phép lũy thừa, công thức là a^b .
/	- Phép chia.
%	- Phép chia lấy phần dư.

3.1. Arithmetic (Toán tử số học)

- Toán tử số học là toán tử dùng để thực hiện các phép toán số học.
- Danh sách các toán tử số học:

Toán tử	Mô tả
++	<ul style="list-style-type: none">- Phép tăng giá trị hiện tại lên 1 đơn vị.- Phép này có hai cách sử dụng đó là đặt nó trước biến (prefix - tiền tố) và đặt nó sau biến (postfix - hậu tố).
--	<ul style="list-style-type: none">- Phép giảm giá trị hiện tại xuống 1 đơn vị.- Phép này cũng có hai cách dùng đó là đặt trước biến (prefix) và đặt sau biến (postfix).

3.2. Assignment (Toán tử gán)

- Toán tử gán dùng để gán giá trị cho một biến.
- Danh sách các toán tử gán:

Toán tử	Ví dụ	Mô tả
=	$a = b$	- Gán giá trị của biến a bằng giá trị của biến b .
+=	$a += b$	- Tương đương với $a = a + b$.
-=	$a -= b$	- Tương đương với $a = a - b$.
*=	$a *= b$	- Tương đương với $a = a * b$.
/=	$a /= b$	- Tương đương với $a = a / b$.
%=	$a \% = b$	- Tương đương với $a = a \% b$.

3.3. Comparison (Toán tử so sánh)

- Toán tử so sánh là toán tử hai ngôi dùng để so sánh giá trị của hai toán hạng với nhau.
- Danh sách các toán tử so sánh:

Toán tử	Ví dụ	Mô tả
$>$	$a > b$	<ul style="list-style-type: none">- Trả về true nếu a lớn hơn b.- Trả về false nếu b lớn hơn a.
$<$	$a < b$	<ul style="list-style-type: none">- Trả về true nếu a nhỏ hơn b.- Trả về false nếu b nhỏ hơn a.
\geq	$a \geq b$	<ul style="list-style-type: none">- Trả về true nếu a lớn hơn hoặc bằng b.- Trả về false nếu a nhỏ hơn b.
\leq	$a \leq b$	<ul style="list-style-type: none">- Trả về true nếu a nhỏ hơn hoặc bằng b.- Trả về false nếu a lớn hơn b.

3.3. Comparison (Toán tử so sánh)

- Toán tử so sánh là toán tử hai ngôi dùng để so sánh giá trị của hai toán hạng với nhau.
- Danh sách các toán tử so sánh:

Toán tử	Ví dụ	Mô tả
<code>==</code>	<code>a == b</code>	<ul style="list-style-type: none">- Trả về true nếu a bằng b.- Trả về false nếu a khác b.- Không nghiêm ngặt.
<code>===</code>	<code>a === b</code>	<ul style="list-style-type: none">- Trả về true nếu giá trị của a bằng giá trị của b và a, b phải cùng kiểu dữ liệu.- Nghiêm ngặt.
<code>!=</code>	<code>a != b</code>	<ul style="list-style-type: none">- Trả về true nếu a khác b.- Trả về false nếu a bằng b.- Không nghiêm ngặt.
<code>!==</code>	<code>a !== b</code>	<ul style="list-style-type: none">- Trả về true nếu giá trị của a khác giá trị của b và a, b phải khác kiểu dữ liệu.- Nghiêm ngặt.

3.4. Logical (Toán tử logic)

- Toán tử logic là toán tử **kết nối** hai hay nhiều **biểu thức**, dùng để kiểm tra mối quan hệ logic giữa các biểu thức.
- Kết quả cuối cùng phụ thuộc vào giá trị của từng biểu thức và loại toán tử logic.
- Danh sách các toán tử logic:

Toán tử	Mô tả
&&	- AND : trả về kết quả là true khi cả hai toán hạng đều true.
 	- OR : trả về kết quả là true khi cả hai hoặc một trong hai toán hạng là true
!	- NOT : Chuyển đổi giá trị của toán hạng từ true sang false hoặc từ false sang true.

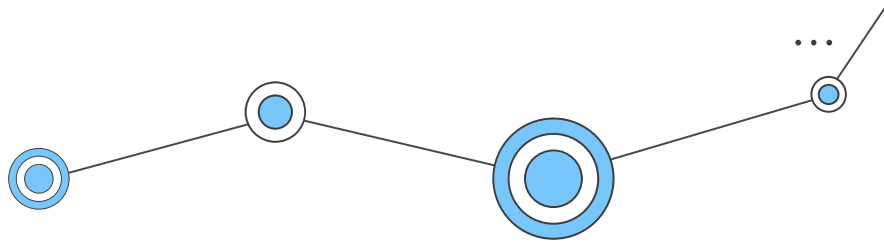
04

Data Types (Kiểu dữ liệu)



4.1. Kiểu dữ liệu nguyên thủy (Primitive Data)

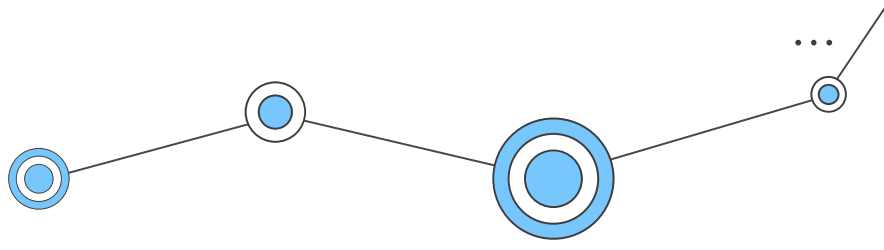


- Khái niệm: **Kiểu dữ liệu nguyên thủy** là kiểu dữ liệu mà **giá trị không thể thay đổi** được.
 - Có **6 kiểu**: Number, String, Boolean, Undefined, Null, Symbol.
 - Kiểu **Number**
 - Là kiểu dữ liệu **dạng số**.
 - Hai loại số là: **số nguyên** và **số thực**.
 - **3 số đặc biệt** là:
 - **Infinity**: là số dương vô cùng.
 - **-Infinity**: là số âm vô cùng.
 - **NaN**: là viết tắt của Not a Number, những trường hợp tính toán sai hoặc kết quả của một phép tính không xác định.
- 



4.1. Kiểu dữ liệu nguyên thủy (Primitive Data)



- Khái niệm: **Kiểu dữ liệu nguyên thủy** là kiểu dữ liệu mà **giá trị không thể thay đổi** được.
 - Có **6 kiểu**: Number, String, Boolean, Undefined, Null, Symbol.
 - Kiểu **String**
 - Là kiểu dữ liệu dùng để **biểu diễn chữ, văn bản, đoạn văn bản,...**
 - Có **3 cách** để biểu diễn string:
 - Dùng dấu nháy đơn: `'`
 - Dùng dấu nháy kép: `"`
 - Dùng dấu backtick: ```
 - Kiểu **Boolean**
 - Là kiểu dữ liệu **logic** chỉ bao gồm 2 giá trị:
 - **true** (đúng, chính xác)
 - **false** (sai, không chính xác)
- 

4.1. Kiểu dữ liệu nguyên thủy (Primitive Data)

- Khái niệm: **Kiểu dữ liệu nguyên thủy** là kiểu dữ liệu mà **giá trị không thể thay đổi** được.
- Có **6 kiểu**: Number, String, Boolean, Undefined, Null, Symbol.
- Kiểu **Undefined**
 - Là kiểu dữ liệu mà khi **khai báo ra một biến** và **không gán giá trị** thì kết quả trả về là undefined.
- Kiểu **Null**
 - Là kiểu dữ liệu đặc biệt, chỉ bao gồm một **giá trị là null** (không biết giá trị, không có giá trị).
- Kiểu **Symbol**
 - Symbol là một kiểu dữ liệu nguyên thủy dùng để tạo ra các **giá trị duy nhất** (unique value) và **bất biến** (immutable).

4.2. Dữ liệu phức tạp (Complex Data)

- Có **2 kiểu**: Function, Object.
- Kiểu **Function**
 - Là một **chương trình con** giúp **thực thi** một **công việc cụ thể**.
 - Cú pháp:

```
function tenHam(thamSo1, thamSo2, ...) {  
  // Code  
}
```

4.2. Dữ liệu phức tạp (Complex Data)

- Có **2 kiểu**: Function, Object.
- Kiểu **Object**
 - Là một tập hợp gồm các **cặp key - value** (khóa - giá trị).
 - **value** có thể là **bất kỳ kiểu dữ liệu nào**.
 - Có **2 loại**: **Object** và **Array**.

```
var tenBien = {  
  key1: "value 1",  
  key2: "value 2",  
  ...  
}
```

Cú pháp Object

```
var tenBien = [  
  "value 1",  
  "value 2",  
  ...  
]
```

Cú pháp Array

05

Một số hàm built-in

05. Một số hàm built-in

- **Hàm built-in** là những hàm được javascript **định nghĩa sẵn**, chúng ra **chỉ việc sử dụng**.
- Hàm **alert()**
 - Có nhiệm vụ **in một thông báo popup**.
 - Nó có một tham số truyền vào và tham số này chính là nội dung thông báo.
 - Cú pháp: **alert(message);**
- Hàm **confirm()**
 - Hiển thị thông báo popup và có thêm **hai lựa chọn** là **Yes** và **No**.
 - Nếu **chọn Yes** thì trả về **TRUE**.
 - Nếu chọn **NO** thì trả về **FALSE**.
 - Nó có một tham số truyền vào và tham số này chính là nội dung thông báo.
 - Cú pháp: **confirm(message);**

05. Một số hàm built-in

- **Hàm built-in** là những hàm được javascript **định nghĩa sẵn**, chúng ra **chỉ việc sử dụng**.
- Hàm **prompt()**
 - Hàm prompt() dùng để **lấy thông tin từ người dùng**.
 - Gồm có **hai tham số** truyền vào là **nội dung thông báo** và **giá trị ban đầu**.
 - Cú pháp: **prompt(title, defaultValue);**
 - Trong đó:
 - **title**: nội dung chữ hiển thị.
 - **defaultValue**: giá trị mặc định cho ô nhập (không bắt buộc).
- Hàm **console**
 - Dùng để **in ra nội dung ở tab console** trên dev tools.
 - Có 3 hàm hay sử dụng là:
 - **console.log()**: in ra **thông báo**
 - **console.warn()**: in ra **cảnh báo**
 - **console.error()**: in ra **lỗi**

05. Một số hàm built-in

- **Hàm built-in** là những hàm được javascript **định nghĩa sẵn**, chúng ra **chỉ việc sử dụng**.
- Hàm **setTimeout()**
 - Code sẽ **chạy 1 lần duy nhất** sau một khoảng thời gian.
 - Cú pháp:

```
setTimeout(function () {  
  // Code  
}, time);
```

05. Một số hàm built-in

- **Hàm built-in** là những hàm được javascript **định nghĩa sẵn**, chúng ra **chỉ việc sử dụng**.
- Hàm **setInterval()**
 - Code sẽ **chạy lặp lại sau một khoảng thời gian**.
 - Cú pháp:

```
setInterval(function () {  
    // Code  
}, time);
```



06

Typeof



06. Typeof

- **typeof** cho phép **trả về kiểu của một biến**.
- Cú pháp: **typeof x** (với **x** là một biến hoặc biểu thức).
- Ví dụ:

```
var a = 10;
var b = "Nội dung...";
var c;
var d = null;
var e = true;

console.log(typeof a); // number
console.log(typeof b); // string
console.log(typeof (a + b)); // string
console.log(typeof c); // undefined
console.log(typeof d); // object
console.log(typeof e); // boolean
console.log(typeof a == "number"); // true
```



07

Làm việc với String

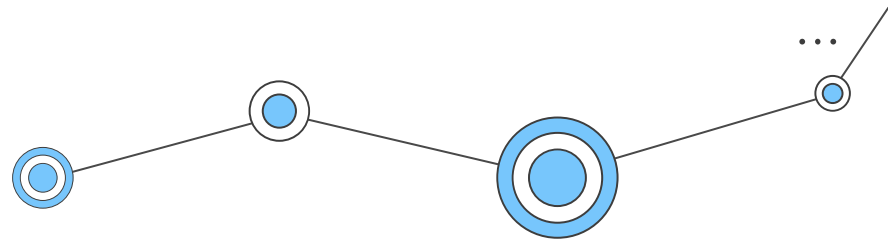


7.1. Length

- Length trả về **độ dài** của chuỗi.
- Ví dụ:

```
var fullName = `Le Van A`;  
console.log(fullName.length);
```

Minh họa



7.2. indexOf()

- **Tìm vị trí đầu tiên** của một chuỗi nằm trong một chuỗi.
- Nếu **không tìm thấy** sẽ trả về **-1**.
- Có phân biệt chữ hoa, chữ thường.
- Cú pháp: **string.indexOf(searchvalue, start)**
- Trong đó:
 - **searchvalue**: Giá trị cần tìm kiếm (bắt buộc).
 - **start**: Vị trí bắt đầu tìm kiếm (tính từ trái qua phải) và sẽ tìm kiếm xuôi tiếp (mặc định bắt đầu từ 0).
- Ví dụ:

Minh họa



```
var myString = `Xin chào! Tôi tên Nam. Tôi năm nay 18 tuổi.`;  
console.log(myString.indexOf("Tôi")); // Trả về 10  
console.log(myString.indexOf("Hải")); // Trả về -1  
console.log(myString.indexOf("Nam")); // Trả về 18  
console.log(myString.indexOf("nam")); // Trả về -1  
console.log(myString.indexOf("Tôi", 11)); // Trả về 23
```

7.3. lastIndexOf()

- **Tìm vị trí cuối cùng** của một chuỗi nằm trong một chuỗi.
- Nếu **không tìm thấy** sẽ trả về **-1**.
- Có phân biệt chữ hoa, chữ thường.
- Cú pháp: **string.lastIndexOf(searchvalue, start)**
- Trong đó:
 - **searchvalue**: Giá trị cần tìm kiếm (bắt buộc).
 - **start**: Vị trí bắt đầu tìm kiếm (tính từ trái qua phải) và sẽ tìm kiếm ngược lại (mặc định bắt đầu từ 0).
- Ví dụ:

Minh họa



```
var myString = `Xin chào! Tôi tên Nam. Tôi năm nay 18 tuổi.`;  
console.log(myString.lastIndexOf("Tôi", 24)); // Trả về 23  
console.log(myString.lastIndexOf("Tôi", 22)); // Trả về 10  
console.log(myString.lastIndexOf("Tôi", 10)); // Trả về 10  
console.log(myString.lastIndexOf("Tôi", 9)); // Trả về -1
```

7.4. slice()

- Dùng để **cắt một chuỗi** và trả về một chuỗi mới.
- Chuỗi ban đầu không thay đổi.
- Cú pháp: **string.slice(start, end)**
- Trong đó:
 - **start**: Vị trí bắt đầu cắt (bắt buộc).
 - **end**: Vị trí kết thúc cắt (không bắt buộc).
- Ví dụ:

Minh họa

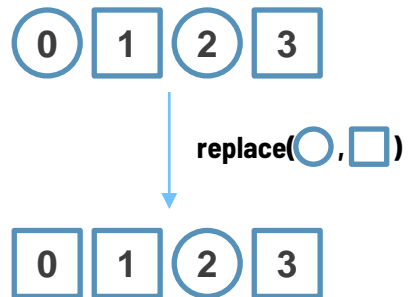


```
var myString = `Xin chào! Tôi tên Nam. Tôi năm nay 18 tuổi.`;  
console.log(myString.slice(10, 21)); // Tôi tên Nam  
console.log(myString.slice(0)); // Lấy toàn bộ chuỗi  
console.log(myString.slice(0, 1)); // X (Lấy ký tự đầu tiên của chuỗi)  
console.log(myString.slice(-1)); // . (Lấy ký tự cuối cùng của chuỗi)  
console.log(myString); // Vẫn giữ nguyên chuỗi gốc
```

7.5. replace()

- Dùng để **thay thế** 1 chuỗi thành 1 chuỗi mới.
- Nhưng chỉ **thay thế chuỗi đầu tiên** mà nó **tìm thấy**.
- Cú pháp: **string.replace(searchValue, newValue)**
- Trong đó:
 - **searchValue**: Từ khóa cần thay thế (bắt buộc).
 - **newValue**: Từ khóa mới để thay thế (bắt buộc).
- Ví dụ:

Minh họa



```
var myString = `Xin chào! Tôi tên Nam. Tôi năm nay 18 tuổi.`;  
console.log(myString.replace("Tôi", "Mình"));  
console.log(myString.replace(/Tôi/g, "Mình"));
```

7.6. toUpperCase() và toLowerCase()

- **toUpperCase()** dùng để **VIẾT HOA TẤT CẢ CÁC CHỮ CÁI**.
- **toLowerCase()** dùng để **viết thường tất cả các chữ cái**.
- Ví dụ:

```
var myString = `Xin chào! Tôi tên Nam. Tôi năm nay 18 tuổi.`;  
console.log(myString.toUpperCase());  
console.log(myString.toLowerCase());
```


7.7. trim()

- Dùng để bỏ đi khoảng trắng ở 2 đầu.
- Các khoảng trắng giữa các ký tự không bị loại bỏ.
- Cú pháp: **string.trim()**
- Ví dụ:

```
var myString = `  Xin chào! Tôi tên Nam.  Tôi năm   nay 18 tuổi.  `;  
console.log(myString);  
console.log(myString.trim());
```

7.8. charAt()

- Dùng để lấy ký tự thông qua index.
- Cú pháp: **string.charAt(index)**
- Với index là vị trí của ký tự cần lấy.
- Ví dụ:

Minh họa



```
var myString = `Đặng Phương Nam`;  
console.log(myString.charAt()); // Lấy ký tự đầu tiên  
console.log(myString.charAt(0)); // Lấy ký tự đầu tiên  
console.log(myString.charAt(1)); // Lấy ký tự thứ 2  
console.log(myString.charAt(myString.length - 1)); // Lấy ký tự cuối cùng
```

7.9. split()

- Chuyển một chuỗi thành một array.
- Nhưng cần tìm ra điểm chung của chuỗi đó.
- Ví dụ:

```
var myString = `HTML5, CSS3, Javascript`;
console.log(myString.split()); // Cả string là 1 phần tử
console.log(myString.split("")); // Mỗi ký tự là 1 phần tử
console.log(myString.split(", ")); // Mỗi từ là 1 phần tử
console.log(myString.split(", ", 2)); // Mỗi từ là 1 phần tử, lấy tối đa 2 phần tử
```



08

Làm việc với Number



8.1. isNaN()

- Dùng để **kiểm tra** một biến có phải là **NaN** không.
- NaN: là viết tắt của Not a Number.
- Cú pháp: **isNaN(tenBiến)**
- Ví dụ:

```
var a = 10;  
var b = "Test";  
var result = a / b;  
console.log(result); // Trả về NaN  
console.log(isNaN(result)); // Trả về true
```

8.2. toString()

- Chuyển kiểu số thành kiểu string.
- Cú pháp: **tenBien.toString()**
- Ví dụ:

```
var a = 10;
var b = a.toString();
var c = (10).toString();
console.log(a); // Trả về số 10
console.log(typeof a); // Trả về kiểu number
console.log(b); // Trả về chuỗi 10
console.log(typeof b); // Trả về kiểu string
console.log(c); // Trả về chuỗi 10
console.log(typeof c); // Trả về kiểu string
```

8.3. toFixed()

- Dùng để **làm tròn số thập phân**.
- Nếu số ngay sau số cần làm tròn
 - **≥ 5** thì sẽ **làm tròn lên**
 - **< 5** sẽ **làm tròn xuống**.
- Cú pháp: **toFixed(digits)**
- Trong đó: **digits** là số chữ số sau dấu thập phân ($0 \leq \text{digits} \leq 100$). Không điền mặc định là 0.
- Ví dụ:

```
var a = 12.3456;  
console.log(a.toFixed()); // Trả về 12  
console.log(a.toFixed(0)); // Trả về 12  
console.log(a.toFixed(1)); // Trả về 12.3  
console.log(a.toFixed(2)); // Trả về 12.35  
console.log(a.toFixed(3)); // Trả về 12.346
```



09

Làm việc với Array



9.1. toString()

- Chuyển array thành string.
- Tự động thêm dấu phẩy ngăn cách.
- Cú pháp: **array.toString()**
- Ví dụ:

```
var list = ["HTML5", "CSS3", "Javascript"];  
console.log(list.toString());  
// Trả về: "HTML5,CSS3,Javascript"
```

Minh họa

["a", "b", "c"] $\xrightarrow{\text{toString()}}$ **"a,b,c"**

9.2. join()

- Chuyển array thành string.
- Thêm dấu bất kỳ để ngăn cách giữa các phần tử.
- Cú pháp: **array.join(separator)**
- Trong đó:
 - **separator** (dải phân cách) là dấu ngăn cách để thêm vào string. Mặc định là dấu phẩy.
- Ví dụ:

Minh họa

`["a", "b", "c"]` $\xrightarrow{\text{join("-")}}$ `"a-b-c"`

```
var list = ["HTML5", "CSS3", "Javascript"];
console.log(list.join()); // Trả về: "HTML5,CSS3,Javascript"
console.log(list.join(",")); // Trả về: "HTML5,CSS3,Javascript"
console.log(list.join("")); // Trả về: "HTML5CSS3Javascript"
console.log(list.join(" ")); // Trả về: "HTML5, CSS3, Javascript"
console.log(list.join(" - ")); // Trả về: "HTML5 - CSS3 - Javascript"
```

9.3. pop()

- Dùng để xóa phần tử cuối mảng.
- Trả về phần tử cuối mảng.
- Cú pháp: **array.pop()**
- Ví dụ:

```
var list = ["HTML5", "CSS3", "Javascript"];  
console.log(list.pop()); // Trả về: "Javascript"  
console.log(list); // Trả về: ["HTML5", "CSS3"]
```

Mình họa

["a", "b", "c"] $\xrightarrow{\text{pop()}}$ **["a", "b"]**

9.4. push()

- Thêm một hoặc nhiều phần tử vào cuối mảng.
- Trả về độ dài mới của mảng.
- Cú pháp: `array.push(item1, item2, ..., itemX)`
- Ví dụ:

Minh họa

`["a", "b", "c"]` $\xrightarrow{\text{push("d")}}$ `["a", "b", "c", "d"]`

```
var list = ["HTML5", "CSS3", "Javascript"];  
console.log(list.push("Bootstrap 4", "ReactJS")); // Trả về: 5  
console.log(list);  
// Trả về: ["HTML5", "CSS3", "Javascript", "Bootstrap 4", "ReactJS"]
```

9.5. shift()

- Dùng để xóa phần tử đầu mảng.
- Trả về phần tử ở đầu mảng.
- Cú pháp: **array.shift()**
- Ví dụ:

```
var list = ["HTML5", "CSS3", "Javascript"];  
console.log(list.shift()); // Trả về: "HTML5"  
console.log(list); // Trả về: ["CSS3", "Javascript"]
```

Minh họa

["a", "b", "c"] $\xrightarrow{\text{shift()}}$ **["b", "c"]**

9.6. unshift()

- Thêm một hoặc nhiều phần tử vào đầu mảng.
- Trả về độ dài mới của mảng.
- Cú pháp: **array.unshift(item1, item2, ..., itemX)**
- Ví dụ:

Minh họa

["a", "b", "c"] $\xrightarrow{\text{unshift("d")}}$ **["d", "a", "b", "c"]**

```
var list = ["HTML5", "CSS3", "Javascript"];  
console.log(list.unshift("Bootstrap 4", "ReactJS")); // Trả về: 5  
console.log(list);  
// Trả về: ["Bootstrap 4", "ReactJS", "HTML5", "CSS3", "Javascript"]
```

9.7. splice()

- Xóa hoặc chèn phần tử mới vào mảng.
- Trả về mảng bị xóa.
- Cú pháp: **array.splice(index, howmany, item1,, itemX)**
- Trong đó:
 - **index**: Vị trí thêm/xóa phần tử (bắt buộc).
 - **howmany**: Số phần tử cần xóa (không bắt buộc).
 - **item1, ..., itemX**: Các phần tử mới được thêm vào (không bắt buộc).
- Ví dụ 1: Chèn phần tử mới vào mảng

```
var list = ["HTML5", "CSS3", "Javascript"];  
console.log(list.splice(2, 0, "Bootstrap 4", "ReactJS")); // Trả về: []  
console.log(list);  
// Trả về: ["HTML5", "CSS3", "Bootstrap 4", "ReactJS", "Javascript"]
```

9.7. splice()

- Xóa hoặc chèn phần tử mới vào mảng.
- Trả về mảng bị xóa.
- Cú pháp: **array.splice(index, howmany, item1,, itemX)**
- Trong đó:
 - **index**: Vị trí thêm/xóa phần tử (bắt buộc).
 - **howmany**: Số phần tử cần xóa (không bắt buộc).
 - **item1, ..., itemX**: Các phần tử mới được thêm vào (không bắt buộc).
- Ví dụ 2: Xóa phần tử trong mảng

```
var list = ["HTML5", "CSS3", "Javascript"];  
console.log(list.splice(1, 1)); // Trả về: ["CSS3"]  
console.log(list); // Trả về: ["HTML5", "Javascript"]
```


9.7. splice()

- Xóa hoặc chèn phần tử mới vào mảng.
- Trả về mảng bị xóa.
- Cú pháp: **array.splice(index, howmany, item1,, itemX)**
- Trong đó:
 - **index**: Vị trí thêm/xóa phần tử (bắt buộc).
 - **howmany**: Số phần tử cần xóa (không bắt buộc).
 - **item1, ..., itemX**: Các phần tử mới được thêm vào (không bắt buộc).
- Ví dụ 3: Xóa phần tử và chèn phần tử mới vào mảng

```
var list = ["HTML5", "CSS3", "Javascript"];
console.log(list.splice(2, 1, "Bootstrap 4", "ReactJS"));
// Trả về: ["Javascript"]
console.log(list);
// Trả về: ["HTML5", "CSS3", "Bootstrap 4", "ReactJS"]
```

9.8. concat()

- Dùng để **nối 2 array**.
- Không làm ảnh hưởng đến mảng ban đầu.
- Cú pháp: **array1.concat(array2, array3, ..., arrayX)**
- Ví dụ:

Minh họa

`["a", "b"]` $\xrightarrow{\text{concat("c")}}$ `["a", "b", "c"]`

```
var list = ["HTML5", "CSS3", "Javascript"];
var list2 = ["Bootstrap 4", "ReactJS"];
console.log(list.concat(list2));
// Trả về: ["HTML5", "CSS3", "Javascript", "Bootstrap 4", "ReactJS"]
console.log(list);
// Trả về: ["HTML5", "CSS3", "Javascript"]
```

9.9. slice()

- Dùng để cắt các phần tử.
- Không làm ảnh hưởng đến mảng ban đầu.
- Cú pháp: **array.slice(start, end)**
- Trong đó:
 - **start**: Vị trí bắt đầu. Mặc định là 0. (không bắt buộc).
 - **end**: Vị trí kết thúc. Mặc định là phần tử cuối cùng. (không bắt buộc).
- Ví dụ:

Minh họa



```
var list = ["HTML5", "CSS3", "Javascript", "Bootstrap 4", "ReactJS"];
console.log(list.slice(3)); // Trả về: ["Bootstrap 4", "ReactJS"]
console.log(list.slice(1, 3)); // Trả về: ["CSS3", "Javascript"]
console.log(list.slice(-3, -1)); // Trả về: ["Javascript", "Bootstrap 4"]
console.log(list);
// Trả về: ["HTML5", "CSS3", "Javascript", "Bootstrap 4", "ReactJS"]
```

Bài tập

- **Link bài tập:** <https://dacavn.notion.site/B-i-t-p-b-i-05-Javascript-c-b-n-Ti-t-1-2bdbb3e576c5454495401933bbe770c0?pvs=4>

