

Front-end Essentials

JavaScript Fundamentals



- Hiểu và nắm vững cấu trúc ngữ pháp của ngôn ngữ lập trình JavaScript;
- Hiểu và khai báo được biến, Phạm vi của biến, biểu thức logic, cấu trúc điều kiện if-else, switch-case, cấu trúc vòng lặp;
- Hiểu, khai báo và sử dụng cấu trúc mảng trong JavaScript để giải quyết các bài toán cụ thể;
- Hiểu cách tạo và truyền tham số vào cho hàm, sử dụng hàm;
- Sử dụng được các hàm có sẵn của JS: Hàm xử lý số, hàm xử lý chuỗi, hàm xử lý ngày tháng, và các hàm liên quan đến biến, kiểu biến;

- Giới thiệu chung về JavaScript (JS)
- Liên kết JS trong trang HTML
- Cú pháp khai báo biến, biểu thức logic
- Cú pháp khai báo và sử dụng cấu trúc điều kiện if-else, switch-case
- Cú pháp khai báo và sử dụng vòng lặp: for, while
- Hàm: Khai báo và cách sử dụng hàm trong JS
- Sự kiện và bắt sự kiện trong JS

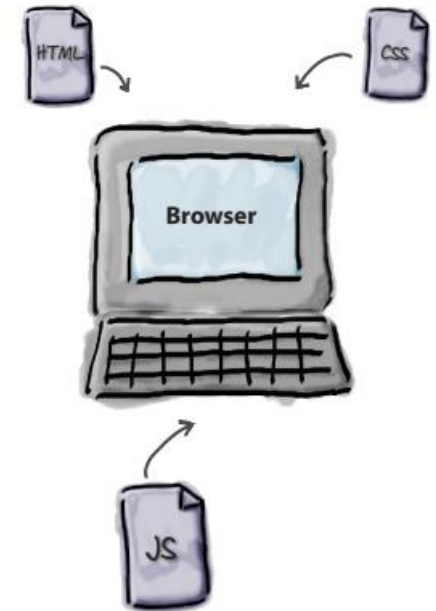
JavaScript Fundamentals

Session 1: JavaScript Basics



Giới thiệu chung về JS

- JavaScript (JS) là một ngôn ngữ lập trình kịch bản được thực thi trên phía trình duyệt của người dung đầu cuối;
- Thường kết hợp với các phần tử HTML để làm cho Web HTML trở nên động hơn và tương tác tốt hơn;
 - ✓ Validation dữ liệu
 - ✓ Hiển thị các hộp thoại
 - ✓ Xử lý sự kiện
 - ✓ Thay đổi style của các phần tử HTML (DOM)
 - ✓ ...
- JS là ngôn ngữ lập trình mở và đa nền tảng



Ưu điểm chung của JS

- Dễ học
- Thực thi bên trình duyệt người dung đầu cuối, để hạn chế việc xử lý bên phía Server;
- Chạy được trên nhiều hệ điều hành khác nhau;
- Sử dụng kết hợp được với nhiều ngôn ngữ lập trình Web khác: Java, ASP.NET, Ruby, Perl, PHP, ...
- Js tốt giúp tăng hiệu suất tải của trang web (load nhanh, nhẹ hơn)
- Giảm request không hợp lệ tới Server
- Được dung để tạo ra nhiều thư viện JS khác: jQuery, Angular, ...

Liên kết JS trong HTML

- Tất cả các trình duyệt đều hiểu và chạy được JS.
- Lập trình Js phải được đặt trong cặp thẻ **<script> ... </script>**
- Cặp thẻ **<script> ... </script>** có thể:
 - ✓ Để **trong hoặc liên kết trong** cặp thẻ **<head> ... </head>** của tài liệu HTML
 - ✓ Hoặc để **trong hoặc liên kết** file Js **trong** cặp thẻ **<body> ... </body>**.Khuyến nghị để cuối cặp thẻ **<body>**



Các trường hợp liên kết JS trong HTML

```
<!DOCTYPE html>
<html>
<head>
  <meta name="viewport" content="width=device-width" />
  <title>JavaScript Demo</title>
  <script>
    //Trường hợp 1: Viết code JavaScript trực tiếp trong này

  </script>
  <!-- Trường hợp 2: Liên kết file JavaScript-->
  <script src="/PathToScriptFile.js"></script>
</head>
<body>
  <h1> JavaScript Tutorials</h1>
  <p>This is JavaScript sample.</p>
  <script>
    //Trường hợp 3: Viết code JavaScript trực tiếp trong này

  </script>
  <!-- Trường hợp 4: Liên kết File JavaScript cuối thẻ Body -->
  <script src="/PathToScriptFile.js"></script>
</body>
</html>
```


Đoạn Chương trình JS đầu tiên

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-
width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <h2>Cùng học JavaScript hôm nay</h2>
  <script>
    alert("Chào mừng bạn đến với JavaScript!!!");
  </script>
</body>
</html>
```

127.0.0.1:5500 says

Chào mừng bạn đến với JavaScript!!!

OK

Hiển thị hộp thoại thông báo

Chú thích trong JS

- Chú thích một dòng dung //
- Chú thích nhiều dòng dung /* Nội dung chú thích */

```
<script>  
    // Đây là dòng chú thích trong JS  
  
    /*  
    Đây là dòng chú thích thứ nhất  
    các dòng tiếp theo ...  
    */  
</script>
```

- Biến trong JS được khai báo bằng từ khóa **var**
- Biến không có kiểu dữ liệu. Js tự xác định kiểu dữ liệu của biến thông qua giá trị của biến
- Tên biến hợp lệ:
 - ✓ Biến được đặt bằng ký tự in hoa hoặc in thường, gạch dưới (_), các chữ số và dấu \$;
 - ✓ Biến đặt phải có ý nghĩa và dễ hiểu;
 - ✓ Biến đặt theo code convention của công ty
- Tên biến **không** hợp lệ:
 - ✓ Bắt đầu bằng số
 - ✓ Trung tên với các từ khóa của Ngôn ngữ JS

```
//Đặt tên biến Đúng  
var firstName;  
var lastName;  
var age;  
var score = 0, highScore = 0, player = '';  
var x, y, z;  
var _money = 0;
```

Các từ khóa trong JS

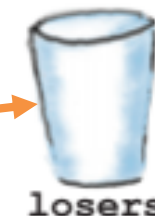
- Các từ khóa trong Js, **việc đặt tên biến tránh các từ khóa này.**

break	delete	for	let	super	void
case	do	function	new	switch	while
catch	else	if	package	this	with
class	enum	implements	private	throw	yield
const	export	import	protected	true	
continue	extends	in	public	try	
debugger	false	instanceof	return	typeof	
default	finally	interface	static	var	

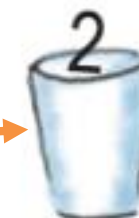


Khai báo biến và gán giá trị cho biến

```
<script>  
  //Khai báo biến  
  var losers;  
  
  //Biến chứa giá trị số nguyên  
  var winners = 2;  
  //Biến chứa giá trị chuỗi  
  var name = "Duke";  
  //Biến chứa giá trị boolean (Đúng/Sai - True/False)  
  var isEligible = false;  
</script>
```



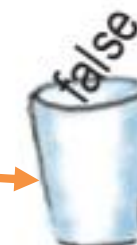
losers



winners



name



isEligible

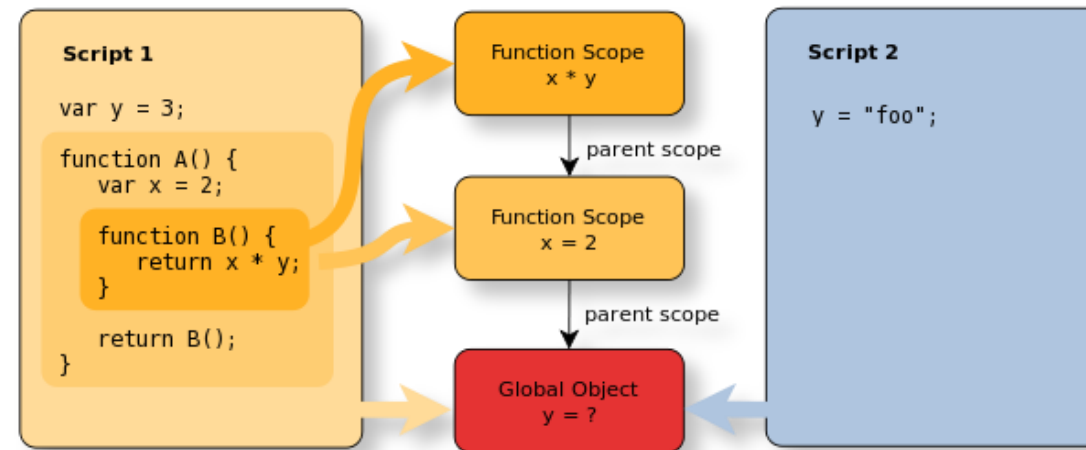
- Gán giá trị của biến thông qua dấu bằng (=)
 - ✓ Sau tên biến ngay khi khai báo;
 - ✓ Hoặc sau khi đã khai báo

```
<script>
    //Đặt tên biến Đúng
    var firstName = 'Fsoft';
    var lastName = 'Academy';
    var age = 0;
    var score = 0, highScore = 0, player = 'Student';
    var x, y, z;
    var _money = 0;

    //Gán giá trị cho biến x, y và z
    x = y = true;
    z = 8.5;
</script>
```

Biến và Phạm vi của biến trong JS

- Trong Js có 02 phạm vi của biến
 - ✓ Biến cục bộ (*Local Variable*): Biến được khai báo bên trong một hàm và chỉ sử dụng được ở trong hàm đó;
 - ✓ Biến toàn cục (*Global Variable*): Biến được khai báo bên ngoài một hàm và có thể được truy cập ở bất kỳ vị trí nào trong chương trình JavaScript



■ Các toán tử trong Js

Toán tử	Mô tả
+	Cộng
-	Trừ
*	Nhân
**	Lũy thừa(theo chuẩn ES2016)
/	Chia
%	Chia lấy phần dư
++	Tăng 1 đơn vị
--	Giảm 1 đơn vị

Toán tử và phép tính	Ví dụ	Tương đương
=	$x = y$	$x = y$
+=	$x += y$	$x = x + y$
-=	$x -= y$	$x = x - y$
*=	$x *= y$	$x = x * y$
/=	$x /= y$	$x = x / y$
%=	$x \% = y$	$x = x \% y$
**=	$x ** = y$	$x = x ** y$

Các toán hạng trong JS

- ✓ Các toán hạng thường dùng trong việc so sánh giữa các biến

Toán hạng	Mô tả
==	So sánh hai biến bằng nhau hay không mà không cần xem xét kiểu dữ liệu của biến
===	So sánh hai biến bằng nhau hay không dựa trên giá trị và kiểu giá trị của chúng
!=	Khác (Không bằng)
!==	Giá trị không bằng nhau hoặc kiểu dữ liệu không giống nhau
>	Lớn hơn
<	Nhỏ hơn
>=	Lớn hơn hoặc bằng
<=	Nhỏ hơn hoặc bằng
?	Toán tử cho biểu thức logic 3 vế. Ví dụ (a>b) ? a : b;

Các toán hạng Logic trong JS

Toán hạng	Mô tả
&&	Toán hạng Logic và
	Toán hạng Logic hoặc
!	Toán hạng Logic phủ định (not)

- Khi khai báo biến trong JS **ko cần khai báo kiểu dữ liệu cho biến**;
- JS sẽ tự xác **định kiểu dữ liệu của biến** thông qua **giá trị của biến**;
- Trong **JS có các kiểu dữ liệu** sau đây:
 - ✓ Số (Number)
 - ✓ Chuỗi (String)
 - ✓ Đúng/Sai (Boolean)
 - ✓ Rỗng/Giá trị rỗng (Null) → Biến được khai báo có giá trị Null hoặc rỗng
 - ✓ Không xác định kiểu (Undefined) → Biến được khai báo nhưng không có giá trị
 - ✓ Đối tượng (Object)
 - ✓ Mảng (Array)
 - ✓ Ngày tháng (Date)

Ví dụ các kiểu dữ liệu của biến

```
<script>
    //Biến kiểu số
    var a = 5, b = 10;
    var sum = a + b;
    //Biến kiểu chuỗi
    var firstName = "Duke";
    var lastName = "Donal";
    //Nối chuỗi --> Dùng dấu Cộng
    var fullName = firstName + ' ' + lastName;
    //Biến chứa giá trị Null
    var player = ''; //Hoặc có giá trị bằng null
    //Biến kiểu Boolean (true/false)
    var isActive = false;
    //Biến kiểu Undefined
    var car;
    //hoặc biến có giá trị là undefined
    car = undefined;
    //Biến kiểu đối tượng - Object Person gồm 03 thuộc tính
    var person = {fName:"John", lName:"Doe", age:50, eyeColor:"blue"};
    //Biến kiểu mảng - Array
    var cars = ["Saab", "Volvo", "BMW"];
</script>
```

Xác định kiểu dữ liệu của biến

- Làm thế nào để xác định một biến bất kỳ có kiểu dữ liệu là gì?

Toán tử	Mô tả
typeof	Trả về kiểu dữ liệu của biến
instanceof	Trả về True (Đúng) nếu biến là một thực thể kiểu đối tượng

```
typeof "John"           // Returns "string"
typeof 3.14              // Returns "number"
typeof NaN              // Returns "number"
typeof false            // Returns "boolean"
typeof [1,2,3,4]         // Returns "object"
typeof {name:'John', age:34} // Returns "object"
typeof new Date()        // Returns "object"
```

Hiển thị giá trị ra Web với JS

- Trong Js có các cách sau đây để hiển thị giá trị ra Web:
 - ✓ Hiển thị ra trang HTML sử dụng **document.write()**
 - ✓ Hiển thị dạng hộp thoại sử dụng **window.alert()** hoặc **alert()**
 - ✓ Hiển thị trên Browser Console sử dụng **console.log()**
 - ✓ Hiển thị ra một phần tử HTML hoặc phần tử có id, class sử dụng **innerHTML**

```
<script>
  //Biến kiểu số
  var a = 5, b = 10;
  var sum = a + b;
  //Hiển thị giá trị ra trang HTML
  document.writeln("a = "+a+" và b = "+b+" (a+b) =" +sum);
  //Biến kiểu chuỗi
  var firstName = "Duke";
  var lastName = "Donal";
  //Nối chuỗi --> Dùng dấu Cộng
  var fullName = firstName + ' ' + lastName;
  //Hiển thị giá trị ra trang HTML
  document.writeln("<br> Fullname:" +fullName);
</script>
```



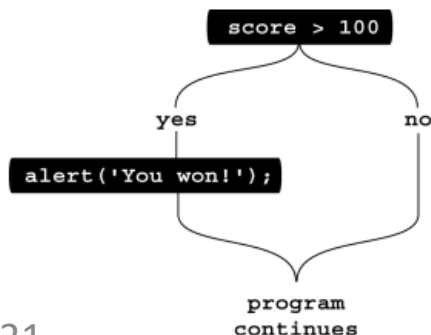
a = 5 và b = 10 (a+b) =15
Fullname:Duke Donal

Cấu trúc điều kiện if-else

```
/* Cấu trúc If đơn
   Chuyện gì xảy ra khi biểu thức điều
   kiện (condition) là True
*/
if (condition) { //true
    //Làm gì đó trong này
}
```

■ Ví dụ (if):

```
var score = 0;
if (score > 10) {
    alert('You Won!');
}
```

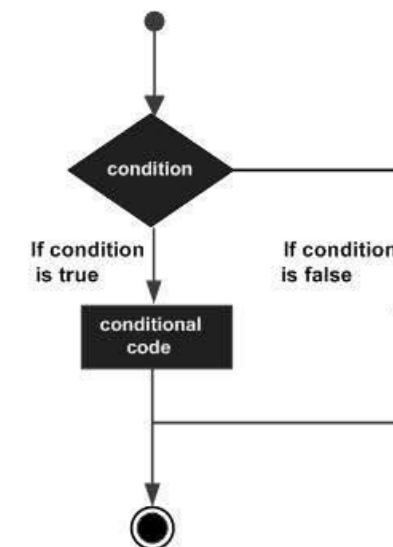


Cấu trúc điều kiện If-else

```
if (condition) {
    // Làm gì đó khi condition là True
} else {
    // Làm gì đó khi condition là False
}
```

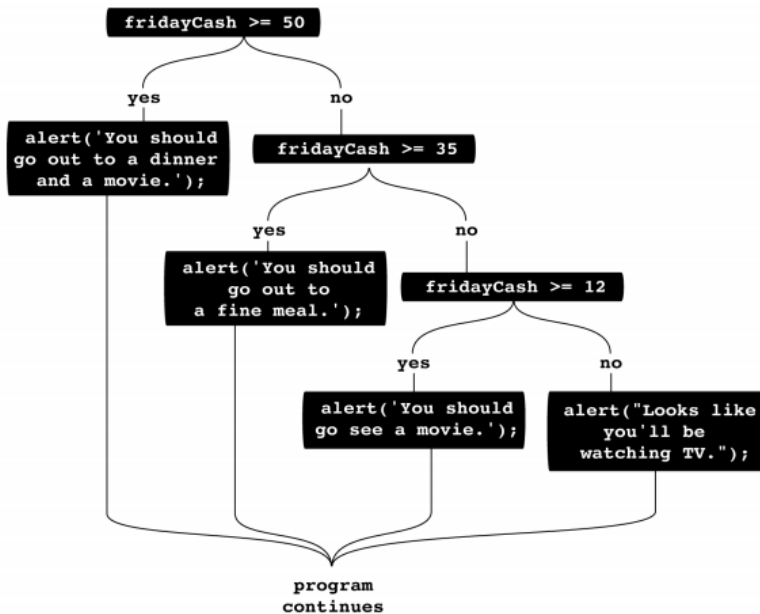
■ Ví dụ (if-else):

```
var mySal = 500;
var yourSal = 800;
if( mySal > yourSal){
    alert("My Salary is greater than your salary");
}else{
    alert("My Salary is less than or equal to your salary");
}
```



Cấu trúc điều kiện if – else - if

- Cấu trúc if-else-if cho phép chúng ta kiểm tra nhiều trường hợp có thể xảy ra đối với biểu thức logic của if



```
if (condition 1) {  
    //Làm gì đó khi biểu thức condition 1 là True  
} else if(condition 2){  
    //Làm gì đó khi biểu thức condition 2 là True  
}else if(condition 3){  
    //Làm gì đó khi biểu thức condition 3 là True  
}else if(condition 4){  
    //Làm gì đó khi biểu thức condition 4 là True  
}else{  
    //Làm gì đó khi không có biểu thức condition nào ở trên là True  
}
```

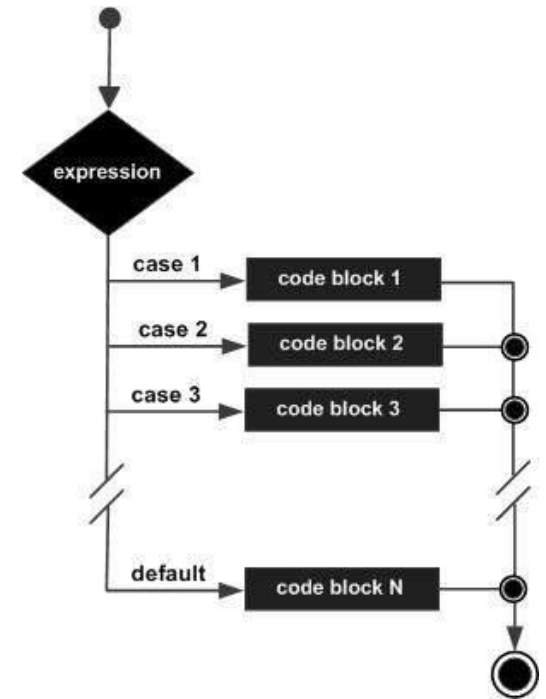
Ví dụ: Hôm nay là thứ Năm, bạn yêu cầu người dùng nhập vào một số tiền (đang có):

- Nếu số tiền đó $\geq 50 \rightarrow$ Hiển thị thông điệp là “Bạn nên đi ăn tối và xem Film”
- $50 < \text{Số tiền} \leq 35 \rightarrow$ Hiển thị thông điệp là “Bạn nên đi mua đồ ăn”
- $35 < \text{Số tiền} \leq 12 \rightarrow$ Hiển thị thông điệp là “Bạn có thể đi xem Film”
- Số tiền không nằm trong các trường hợp trên thì hiển thị thông điệp “Bạn nên ở nhà xem Tivi”

Cấu trúc switch-case

- Muốn kiểm tra giá trị của một biến/Biểu thức nào đó có thể xảy ra để có hành vi/thực hiện phù hợp → Dùng switch-case

```
switch (expression) {  
    case value-1:  
        //Làm gì đó khi giá trị của expression là value-1  
        break; //hoàn thành xử lý khi giá trị của expression là value-1 và kết thúc  
    case value-2:  
        //Làm gì đó khi giá trị của expression là value-2  
        break; //Hoàn thành xử lý khi giá trị của expression là value-2 và kết thúc  
    case value-n:  
        //Làm gì đó khi giá trị của expression là value-n  
        break; //Hoàn thành xử lý khi giá trị của expression là value-1 và kết thúc  
    default:  
        //Làm gì đó khi giá trị của expression KHÔNG thuộc các giá trị ở trên  
}
```



- Expression có thể là một biến hoặc một biểu thức mà kết quả của nó là một giá trị
- Các case chứa giá trị (Số, chuỗi) có thể xảy ra với expression
- Từ khóa break → Kết thúc việc thực thi trong khối case và kết thúc switch-case
- Khối code default sẽ được thực thi khi expression không có giá trị nào thỏa mãn các khối case

Cấu trúc switch-case (tt)

```
<script>
  /*
    Kiểm tra xem hôm nay (Hiện tại) là thứ mấy - Dùng hàm Date()
    và hiển thị ra thông điệp tương ứng với các ngày
  */
  var today = new Date();
  //Dùng hàm getDay() để lấy giá trị ngày trong tuần (0-6)
  switch (today.getDay()) {
    case 0:
      document.write("Hôm nay là Chủ nhật ... Chúc các bạn cuối tuần vui vẻ");
      break;
    case 1:
      document.write("Hôm nay là Thứ 2 ... Thứ 2 là ngày đầu tuần, Bé hứa cố gắng và chăm ngoan");
      break;
    case 2:
      document.write("Hôm nay là Thứ 3 ... ");
      break;
    case 3:
      document.write("Hôm nay là Thứ 4 ... ");
      break;
    case 4:
      document.write("Hôm nay là Thứ 5 ... ");
      break;
    case 5:
      document.write("Hôm nay là Thứ 6 ... ");
      break;
    default:
      document.write("Chỉ có thể là thứ 7 ... Hôm nay là ngày cuối tuần");
      break;
  }
</script>
```



← → ↻ ⓘ 127.0.0.1:5500/switch-case.html

Hôm nay là Thứ 4 ...

Cấu trúc switch-case (tt)

- Kết quả của đoạn code dưới đây là gì? Vì sao?

```
<script>
  var grade = 'A';
  document.write("Entering switch block<br />");
  switch (grade) {
    case 'A': document.write("Good job<br />");
    case 'B': document.write("Pretty good<br />");
    case 'C': document.write("Passed<br />");
    case 'D': document.write("Not so good<br />");
    case 'F': document.write("Failed<br />");
    default: document.write("Unknown grade<br />")
  }
  document.write("Exiting switch block");
</script>
```



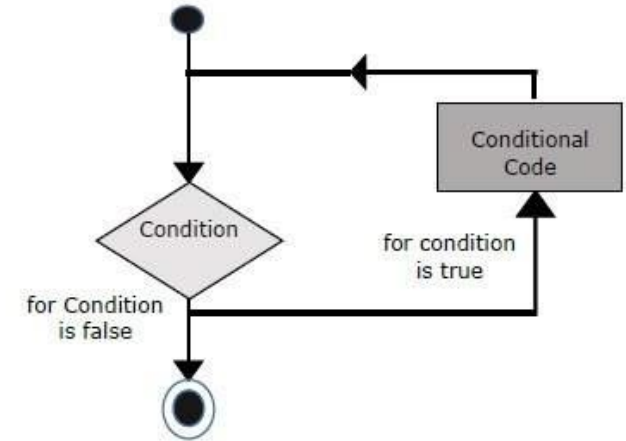
Cấu trúc vòng lặp for

■ Cú pháp


```
for (statement 1; statement 2; statement 3) {  
    // code block to be executed  
}
```

■ Trong đó:

- ✓ **Statement 1** là **biến** xác định **giá trị xuất phát** của vòng lặp
- ✓ **Statement 2** là **biểu thức** xác định **điều kiện để** vòng lặp tiếp tục thực hiện
- ✓ **Statement 3** là xác định sự **tăng/giảm của Statement 1** sau mỗi lần thực hiện của vòng lặp



```
<script>  
    document.write("Starting Loop" + "<br />");  
    for(var count = 0; count < 10; count++) {  
        document.write("Current Count : " + count );  
        document.write("<br />");  
    }  
    document.write("Loop stopped!");  
</script>
```



← → ↻ ⓘ 127.0.0.1:5500/for.html

Starting Loop
Current Count : 0
Current Count : 1
Current Count : 2
Current Count : 3
Current Count : 4
Current Count : 5
Current Count : 6
Current Count : 7
Current Count : 8
Current Count : 9
Loop stopped!

Cấu trúc vòng lặp For-in

- For-in được sử dụng để chạy vòng lặp qua các thuộc tính của một đối tượng (Object)

- Cú pháp:

```
var objProperty ;  
for (objProperty in object) {  
    //statement or block to execute  
}
```

- Trong đó:

- ✓ objProperty là **tên biến** của vòng lặp để **chứa các thuộc tính** của **đối tượng** Object
- ✓ Object là **đối tượng** chúng ta cần duyệt, xử lý

```
<script>  
    //Khai báo đối tượng Person chứa các thuộc tính và giá trị của các thuộc tính: fname, lname và age  
    var Person = {fname:"John", lname:"Doe", age:25};  
    //duyet các thuộc tính của Object  
    var personProperty ;  
    for (personProperty in Person) {  
        document.write("<br>");  
        document.write("Thuộc tính của Object:"+personProperty);  
        document.write("<br>");  
        document.write("Giá trị của thuộc tính:"+personProperty+ " là:" + Person[personProperty]);  
    }  
</script>
```

Thuộc tính của Object:fname
Giá trị của thuộc tính:fname là:John
Thuộc tính của Object:lname
Giá trị của thuộc tính:lname là:Doe
Thuộc tính của Object:age
Giá trị của thuộc tính:age là:25

Cấu trúc vòng lặp For-of

- For-of được sử dụng để chạy vòng lặp qua các giá trị của một đối tượng (Object);
- Vòng lặp for-of này thường thực hiện trên các Object: String, Array, Maps, Set, NodeLists, ...

```
for (variable of iterable) {  
    // code block to be executed  
}
```

- Trong đó:
 - ✓ variable là tên biến của vòng lặp và cũng là một thực thể chứa giá trị của đối tượng iterable
 - ✓ iterable là đối tượng chứa các giá trị

```
var objCars = ['BMW', 'Volvo', 'Mini'];  
for (var x of objCars) {  
    document.write(x + "<br >");  
}
```

```
var objString = 'Learn JavaScript';  
for (var value of objString) {  
    console.log(value);  
}
```



Vòng lặp While và do-while

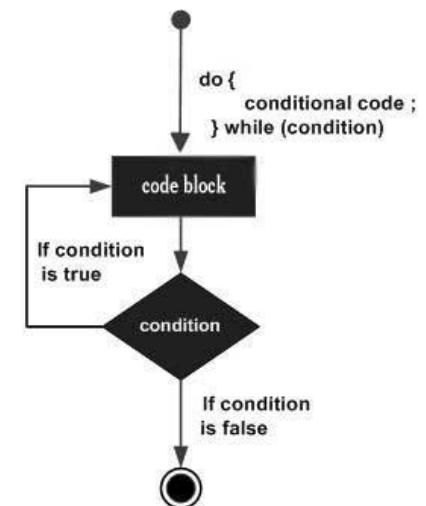
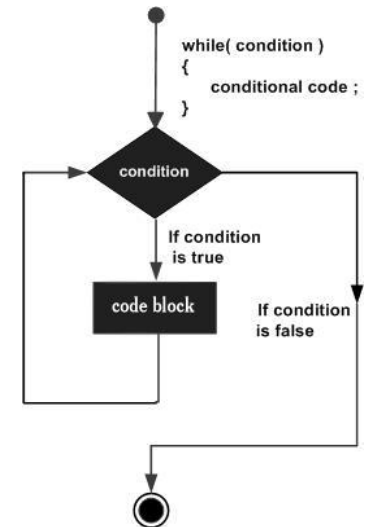
- Vòng lặp **while** sẽ **thực hiện** các **lệnh bên trong vòng lặp** trong **khí điều kiện vòng lặp (condition)** vẫn còn **true**

- Cú pháp:

```
while (condition) {  
    // code block to be executed  
}
```

- Vòng lặp **do-while** sẽ thực hiện các lệnh bên trong vòng lặp một lần rồi sau đó mới kiểm tra điều kiện đầu vào (**condition**) của vòng lặp có còn true hay không? (còn true thì tiếp tục thực hiện)

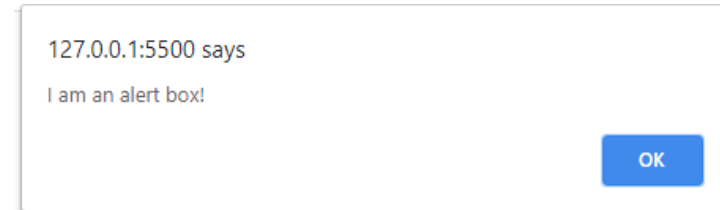
```
do {  
    // code block to be executed  
}  
while (condition);
```



Hiển thị hộp thoại trong Js

- **Alert Box** → Hiển thị một Popup chứa thông điệp, chỉ có nút **Ok**;

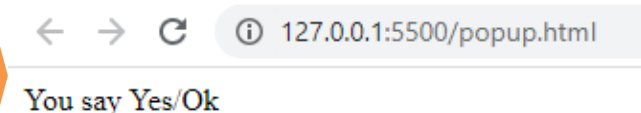
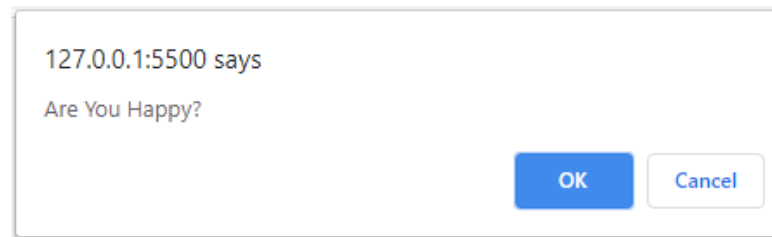
```
alert("I am an alert box!");
```



- **Confirm Box** → Hiển thị hộp thoại chứa thông điệp và có 02 nút **Ok** và **Cancel**

- ✓ Nếu Click vào **Ok** thì kết quả trả về của popup box là **True**
- ✓ Nếu Click vào **Cancel** thì kết quả trả về của popup box là **False**

```
if (confirm("Are You Happy?")) {  
    document.write("You say Yes/Ok");  
} else {  
    document.write("You say No/Cancel");  
}
```

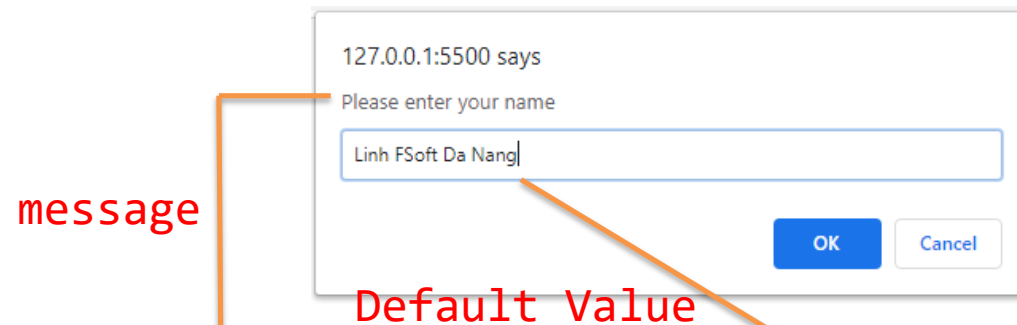


Hiển thị hộp thoại trong Js

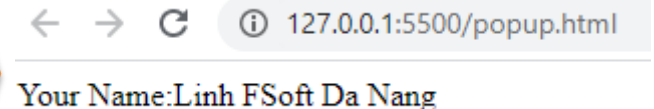
- Prompt Box → Hộp thoại có thông điệp và yêu cầu người dùng nhập vào giá trị trước khi Enter/Ok hoặc Cancel

```
prompt([string message], [string defaultValue]);
```

- Ví dụ:



```
var name = prompt("Please enter your name", "FSoft Da Nang");  
document.write("Your Name:"+name);
```



JavaScript Fundamentals

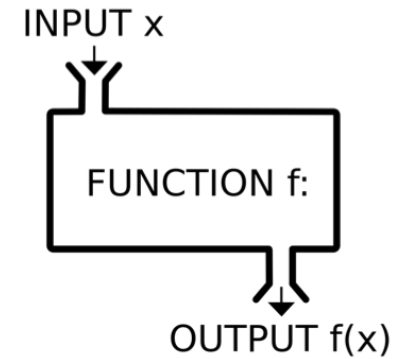
Session 2: *Function - Hàm*



- Giảm việc lập trình những đoạn code giải quyết cùng vấn đề (chỉ khác về giá trị đầu vào) được lặp đi lặp lại
 - ✓ Giảm thời gian lập trình
 - ✓ Tái sử dụng được code đã giải quyết vấn đề
- Một bài toán nhiều vấn đề:
 - ✓ Chia nhỏ bài toán thành từng vấn đề nhỏ
 - ✓ Giảm độ phức tạp trong code
 - ✓ Đơn giản trong việc kiểm tra code
 - ✓ Dễ bảo trì, thay đổi code

Cú pháp Hàm trong JS

```
function name(parameter1, parameter2, parameter3, ...) {  
    // Viết code thực hiện giải quyết vấn đề trong này  
}
```



- Trong đó:
 - ✓ **function**: Là từ khóa trong JS để xác định cái chúng ta sẽ định nghĩa là hàm trong JS
 - ✓ **name**: Tên của hàm chúng ta sẽ đặt. Tên của hàm tuân thủ theo nguyên tắc đặt tên biến trong JS;
 - ✓ **parameter1, parameter2, parameter3** là các tham số đầu vào của bài toán mà chúng ta muốn xử lý trong hàm – Biến cục bộ chỉ sử dụng trong hàm;

Các loại hàm trong JS

- Hàm trả về giá trị - có return giá trị, ví dụ:

```
function addTwoNumber(a, b){  
    return (a+b);  
}
```

- Hàm không trả về giá trị

```
function addTwonumber2(a, b){  
    document.write('Kết quả:'+(a+b));  
}
```

- Hàm anonymous – Hàm không cần đặt tên

```
var res = function(a,b){  
    document.write("Kết quả của hàm:"+ (a+b));  
}
```

- Hàm Closure - Hàm được tạo ra từ bên trong một hàm khác

Khai báo hàm và gọi hàm

- Khai báo hàm – Ví dụ:

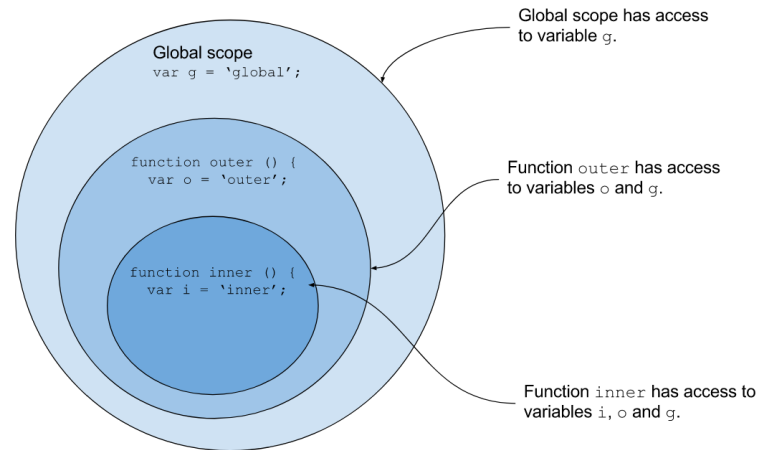
```
/*Hàm tính tổng hai số:  
- Input: 2 số (giả sử 2 biến là a và b)  
- Output: giá trị tổng của 2 số đó  
*/  
function addTwoNumber(a, b){  
    return (a+b);  
}
```

- Gọi hàm (Nếu hàm có tham số thì phải truyền giá trị thực vào cho hàm)

```
// Gọi hàm và truyền giá trị vào cho hàm  
var result = addTwoNumber(5,8);  
//Hiển thị kết quả ra Web  
document.write("Kết quả:"+result);
```

■ *Hàm Closure:*

- ✓ Một hàm được viết lồng vào bên trong một hàm khác (hàm cha - inner function)
- ✓ Nó có thể sử dụng biến toàn cục, biến cục bộ của hàm cha và biến cục bộ của chính nó (lexical scoping)



Khai báo hàm và gọi hàm

```
//Hàm trả về kết quả
function addTwoNumber(a, b){
    return (a+b);
}
//Hàm không trả về kết quả
function addTwonumber2(a, b){
    document.write('Kết quả:'+(a+b));
}
//Hàm không có tên
var res = function(a,b){
    document.write("Kết quả của hàm:"+ (a+b));
}
//Hàm Closure
function sumTwoNumber(a){
    console.log("a = "+a);
    return function (b){
        console.log("b = "+b);
        return (a+b);
    }
}
```

```
// Gọi hàm và truyền giá trị vào cho hàm
var result = addTwoNumber(5,8);
document.write("Kết quả:"+result);

//Gọi hàm không trả về kết quả
addTwonumber2(20,5);

//Gọi hàm không tên
res(4,6);

//Gọi hàm Closure
document.write("Closure -
    Tổng hai số:"+sumTwoNumber(10)(15));
```

← → ↺ ⓘ 127.0.0.1:5500/func.html

Kết quả:13
Kết quả:25
Kết quả của hàm:10
Closure - Tổng hai số:25

JavaScript Fundamentals

Session 3: Array



JavaScript - Array

- Array dùng để **lưu nhiều giá trị** trong **cùng một biến**;
- Mỗi giá trị được lưu vào một vị trí trong mảng
 - ✓ **Vị trí đầu tiên** của mảng có giá trị là **0**

	myCar	Name of the array
0	Chev	Data
1	Ford	
2	Buick	
3	Lincoln	
4	Truck	
		Index number

Comparison of an array to a column of data

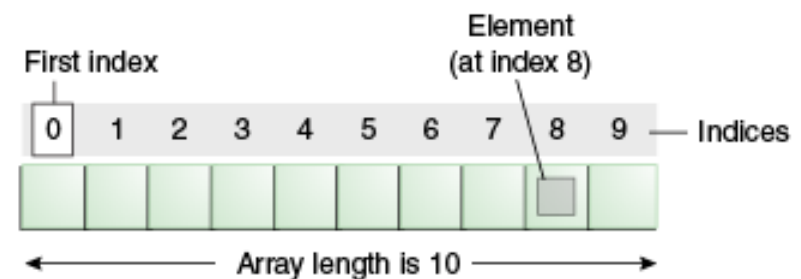
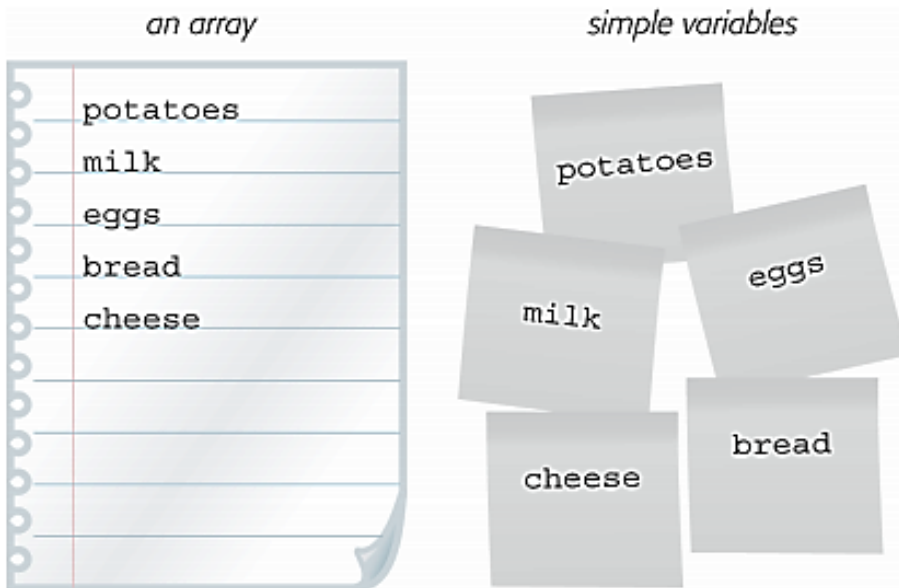
Value	12	"Hello"	"world"	34	90	2.3	2	3	87
Index	0	1	2	3	4	5	6	7	8

- Ví dụ: `var myCar = ["Chev", "Ford", "Buick", "Lincoln", "Truck"];`

JavaScript - Array

```
var days = ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat', 'Sun'];  
alert(days[0]);
```

INDEX VALUE	ITEM	TO ACCESS ITEM
0	Mon	days[0]
1	Tues	days[1]
2	Wed	days[2]
3	Thurs	days[3]
4	Fri	days[4]
5	Sat	days[5]
6	Sun	days[6]



Khai báo mảng trong Js

- Cách 1: Khai báo cấu trúc mảng truyền thông:

```
var array_name = [item1, item2, ...];
```

- Cách 2: Thông qua Đối tượng Array (Phương thức khởi tạo):

```
var array_name = new Array(item1,item2,...);
```

- Ví dụ:

```
var cars = ["Saab", "Volvo", "BMW"];
```

Hoặc

```
var cars = new Array("Saab", "Volvo", "BMW");
```

```
var arr_nums = [12.30, 2.45, 5.00, 7.15, 9.30];
```

Hoặc

```
var arr_nums = new Array(12.30, 2.45, 5.00, 7.15, 9.30);
```

Thuộc tính của Mảng và duyệt mảng

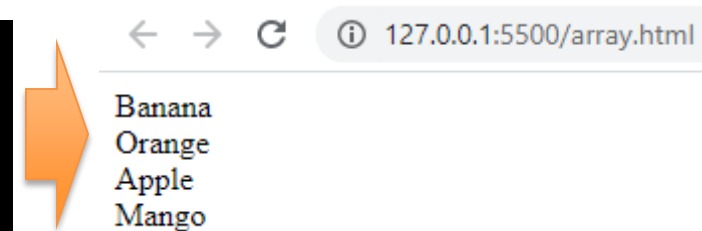
- Độ dài của mảng - **length**

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.length;    // Độ dài của mảng là 4
```

- Duyệt mảng → dùng vòng lặp

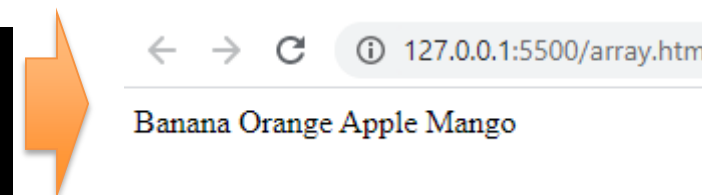
- ✓ Dùng vòng lặp **for**:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];  
for (var index = 0; index < fruits.length; index++) {  
    document.write(fruits[index]+"<br>");  
}
```



- ✓ Dùng vòng lặp **forEach**:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.forEach(  
    item => document.write(item+" ")  
);
```



Ví dụ: Cho mảng số nguyên, hãy:

- ✓ Tính tổng các giá trị trong mảng
- ✓ Đếm số lượng phần tử có giá trị lẻ
- ✓ Đếm số lượng phần tử có giá trị chẵn

```
var numbers = [5,3,10,7,20,55,13];  
var sum = 0, chan = 0, le = 0;  
numbers.forEach(  
    num => {  
        if (num % 2 == 0) {  
            chan++;  
        } else {  
            le++;  
        }  
        sum += num;  
    }  
);  
document.write("Tổng giá trị trong mảng:"+sum);  
document.write("<br>Số lượng phần tử có giá trị chẵn trong mảng:"+chan);  
document.write("<br>Số lượng phần tử có giá trị trong mảng:"+le);
```



← → ↻ ⓘ 127.0.0.1:5500/array.html

Tổng giá trị trong mảng:113
Số lượng phần tử có giá trị chẵn trong mảng:2
Số lượng phần tử có giá trị trong mảng:5

Các hàm/Phương thức xử lý dữ liệu trong mảng

Tên hàm	Chức năng của hàm
<code>pop()</code>	Lấy/xóa một phần tử ở cuối mảng
<code>push(item1, item2, ...)</code>	Thêm các phần tử (item) vào cuối mảng
<code>toString()</code>	Convert mảng về thành một chuỗi, các giá trị trong mảng sẽ được phân cách nhau bằng dấu phẩy
<code>join(separator)</code>	Nối tất cả các phần tử trong mảng thành một chuỗi được phân biệt với nhau bằng một ký hiệu nào đó (separator)
<code>splice(start, deleteCount, item 1, item 2....)</code>	Chèn và xóa phần tử trong mảng: start là vị trí bắt đầu thực hiện, deleteCount số lượng phần tử sẽ bị xóa (0 – không xóa), item 1 , item 2 là các giá trị được thêm vào
<code>sort()</code>	Sắp xếp các phần tử trong mảng
<code>shift()</code>	Lấy/xóa một phần tử ở đầu mảng
<code>unshift(item1, item2, ...)</code>	Thêm phần tử (item) vào đầu mảng
<code>find(logic)</code>	Trả về giá trị đầu tiên tìm thấy (thỏa mãn biểu thức điều kiện)
<code>concat(otherArray)</code>	Nối otherArray vào array đang có
Tham khảo thêm: https://www.w3schools.com/js/js_array_methods.asp	

Array – Ví dụ

```
var numbers = [5,3,10,7,20,55,13];  
//Sắp xếp mảng  
numbers.sort((a,b) => (a - b));  
document.write("Mảng đã sắp xếp:"+ numbers);  
document.write("<br>");  
//Nối thêm giá trị ở mảng khác vào mảng đã có  
var otherArrayNumbers = [10,15,3,9];  
var newNumbers = numbers.concat(otherArrayNumbers);  
document.write("Mảng cũ sau khi được nối thêm:"+newNumbers);  
document.write("<br>");  
//Lấy phần tử ở đầu mảng  
document.write("Phần tử ở đầu mảng:"+newNumbers.shift());  
document.write("<br>");  
//Lấy phần tử ở cuối mảng  
document.write("Phần tử ở cuối mảng:"+newNumbers.pop());  
document.write("<br>");  
//Thêm phần tử ở vị trí bất kỳ mà ko xóa phần tử đã có  
var addingArrayNumber = [100, 150, 50];  
addingArrayNumber.forEach(element => {  
    newNumbers.splice(2,0,element);  
});  
document.write("Mảng chứa giá trị mới:"+newNumbers);  
document.write("<br>");
```

Mảng đã sắp xếp:3,5,7,10,13,20,55
Mảng cũ sau khi được nối thêm:3,5,7,10,13,20,55,10,15,3,9
Phần tử ở đầu mảng:3
Phần tử ở cuối mảng:9
Mảng chứa giá trị mới:5,7,50,150,100,10,13,20,55,10,15,3

JavaScript Fundamentals

Session 4: Một số hàm thao tác trên String, Number, Date, Math



Các hàm xử lý chuỗi (String) trong JS

Chuỗi (String)

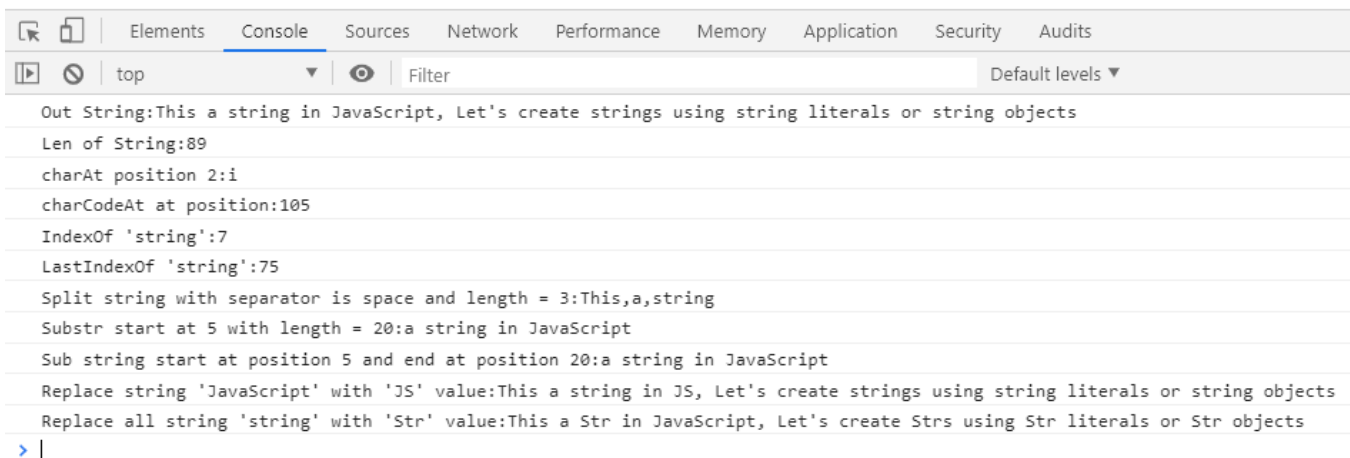
H o m e s w e e t h o m e
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Chỉ số (index) vị trí của các ký tự (char) trong chuỗi (String)

Tên hàm	Mô tả hàm
charAt(index)	Trả về ký tự tại vị trí index
charCodeAt(index)	Trả về giá trị của ký tự tại vị trí index trong bảng mã ASCII
indexOf(string,[index])	Trả về vị trí đầu tiên mà string xuất hiện trong chuỗi từ vị trí index trở đi
lastIndexOf(string,[index])	Trả về vị trí cuối cùng mà string xuất hiện trong chuỗi từ vị trí index trở đi
split([separator],[limit])	Tách các chuỗi trong chuỗi ban đầu thông qua một ký tự đặc biệt separator nào đó, có thể giới hạn số lượng chuỗi sẽ tách limit
substr(start, length)	Tách một chuỗi từ vị trí bắt đầu start với số ký tự length cần tách tiếp
substring(start, end)	Tách một chuỗi từ vị trí bắt đầu start và vị trí kết thúc end
Tham khảo thêm: https://www.w3schools.com/js/js_string_methods.asp	

Ví dụ xử lý chuỗi (String)

```
<script>
  var outString = "This a string in JavaScript, Let's create strings using string literals or string objects";
  console.log("Out String:"+outString);
  console.log("Len of String:"+outString.length);
  console.log("charAt position 2:"+outString.charAt(2));
  console.log("charCodeAt at position:"+outString.charCodeAt(2));
  console.log("IndexOf 'string':"+outString.indexOf("string"));
  console.log("LastIndexOf 'string':"+outString.lastIndexOf("string"));
  console.log("Split string with separator is space and length = 3:"+outString.split(" ",3));
  console.log("Substr start at 5 with length = 20:"+outString.substr(5,22));
  console.log("Sub string start at position 5 and end at position 20:"+outString.substring(5,27));
  console.log("Replace string 'JavaScript' with 'JS' value:"+outString.replace('JavaScript','JS'));
  console.log("Replace all string 'string' with 'Str' value:"+outString.split('string').join('Str'));
</script>
```



Các hàm xử lý số (Number) trong JS

Tên hàm	Mô tả hàm
Number(numObj)	Convert một đối tượng numObj chứa số về giá trị số
parseFloat(num)	Convert một chuỗi số num về giá trị số thực
parseInt(num)	Convert một chuỗi số num về giá trị số nguyên
toFixed(n)	Trả về một số thực có n số lẻ
toPrecision(n)	Trả về một số có độ dài n số
toString()	Chuyển số thành chuỗi số
isFinite()	Trả về một số được kiểm tra có phải là số hữu hạn hay không
isInteger()	Trả về một số được kiểm tra có phải là số nguyên hay không
isNaN()	Kiểm tra một số được đưa vào có là NaN (Không phải là số) hay không

Tham khảo thêm: https://www.w3schools.com/js/js_number_methods.asp

Hàm xử lý Ngày tháng (Date) trong JS

■ Các cách khai báo kiểu ngày tháng trong JS

- ✓ `new Date()` → lấy ngày tháng hiện tại
- ✓ `new Date(year, month, day, hours, minutes, seconds, milliseconds)` → Tạo ngày tháng năm, giờ, phút giây, mili giây theo giá trị được truyền vào
- ✓ `new Date(milliseconds)`
- ✓ `new Date(date string)` → Tạo ngày tháng năm theo chuỗi ngày tháng năm được truyền vào (date string)

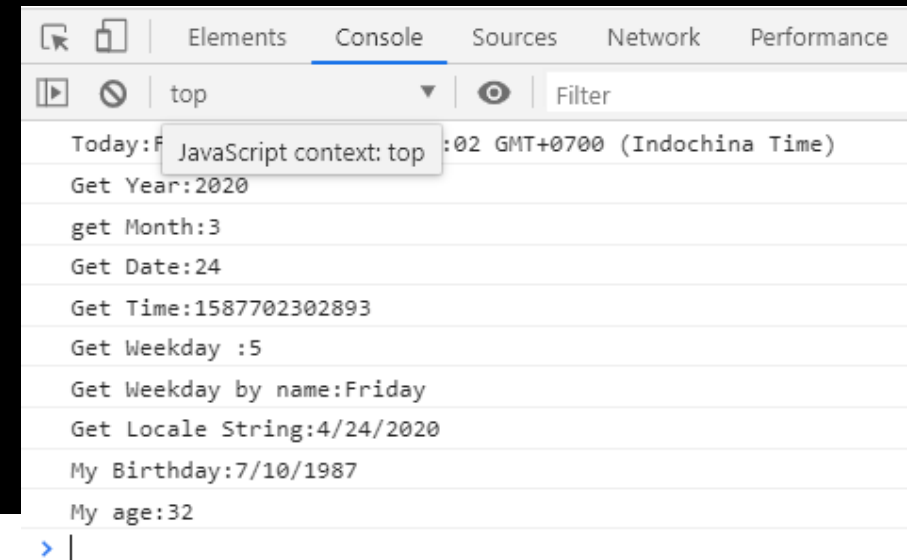
Tên hàm	Mô tả hàm	Tên hàm	
<code>getFullYear()</code>	Trả về năm có 4 số (yyyy)	<code>setFullYear()</code>	Thiết lập lại giá trị năm (4 số)
<code>getMonth()</code>	Trả về tháng (0-11)	<code>setMonth()</code>	Thiết lập lại giá trị tháng (0-11)
<code>getDate()</code>	Trả về ngày (1-31)	<code>setDate()</code>	Thiết lập lại giá trị ngày (1-31)
<code>getHours()</code>	Trả về giờ (0-23)	<code>setHours()</code>	Thiết lập lại giá trị giờ (0-23)
<code>getMinutes()</code>	Trả về phút (0-59)	<code>setMinutes()</code>	Thiết lập lại giá trị phút (0-59)
<code>getDay()</code>	Trả về ngày trong tuần dưới dạng số (0-6), 0 → Chủ nhật	<code>setDay()</code>	Thiết lập lại giá trị ngày trong tuần (0-6)

Tham khảo thêm: https://www.w3schools.com/js/js_dates.asp

Ví dụ xử lý Ngày tháng trong JS

```
<script>
  var today = new Date();
  console.log("Today:"+today);
  console.log("Get Year:"+today.getFullYear());
  console.log("get Month:"+today.getMonth());
  console.log("Get Date:"+today.getDate());
  console.log("Get Time:"+today.getTime());
  console.log("Get Weekday :"+today.getDay());
  var days = ["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"];
  console.log("Get Weekday by name:"+days[today.getDay()]);
  var localeDay = today.toLocaleDateString();
  console.log("Get Locale String:"+localeDay);

  var birthDate = new Date('1987/07/10');
  console.log("My Birthday:" + birthDate.toLocaleDateString());
  var age = today.getFullYear() - birthDate.getFullYear();
  var m = today.getMonth() - birthDate.getMonth();
  if (m < 0 || (m === 0 && today.getDate() < birthDate.getDate())) {
    age--;
  }
  console.log("My age:"+age);
</script>
```



Các hàm toán học (Math)

Hàm	Mô tả
<code>round(num)</code>	Trả về giá trị num đã được làm tròn (Về giá trị số nguyên gần nó nhất)
<code>pow(x,y)</code>	Trả về giá trị của x mũ y
<code>sqrt(x)</code>	Trả về giá trị căn bậc hai của x
<code>abs(x)</code>	Trả về giá trị tuyệt đối của x
<code>ceil(x)</code>	Trả về giá trị x được làm tròn lên số nguyên gần nó nhất
<code>floor(x)</code>	Trả về giá trị x được làm tròn xuống số nguyên gần nó nhất
<code>min(list_num)</code>	Trả về giá trị nhỏ nhất trong danh sách các số (list_num)
<code>max(list_num)</code>	Trả về giá trị lớn nhất trong danh sách các số (list_num)
<code>random()</code>	Trả về một giá trị số thực ngẫu nhiên nằm trong khoảng [0-1]
Tham khảo thêm: https://www.w3schools.com/js/js_math.asp	

JavaScript Fundamentals

Session 5: Object & Array of Objects



- Objects trong JavaScript, cũng tương tự như những ngôn ngữ khác, có thể so sánh như đối tượng trong đời thường
- Trong JavaScript, một object là một thực thể độc lập, với thuộc tính và kiểu.

```
const object = {  
  hello: 'world'  
}
```

PROPERTY VALUE

PROPERTY NAME (KEY)



- Đối tượng trong Javascript là một tập hợp các cặp **khóa - giá trị**
 - ✓ Đối tượng là một tập hợp các thuộc tính
 - ✓ Thuộc tính là một cặp **khóa - giá trị** chứa tên và giá trị
 - ✓ Tên thuộc tính là một giá trị duy nhất **có thể bị ép buộc vào một chuỗi và trở đến một giá trị**
 - ✓ Giá trị thuộc tính có thể là bất kỳ giá trị nào, bao gồm các đối tượng khác hoặc các hàm, được liên kết với tên/khóa

- Các bạn có 3 cách để khai báo đối tượng trong Javascript:
 - ✓ Sử dụng từ khóa `{ }`
 - ✓ Sử dụng từ khóa `new Object()`
 - ✓ Sử dụng phương thức `static`

```
// literal
const dog = { }

// constructor
const cat = new Object();

// static method
const horse = Object.create({ })
```

Phương thức trong Object

- Một đối tượng ngoài các thuộc tính ra nó còn chứa hàm gọi là phương thức.
- Một phương thức là một hàm liên kết với một object, hoặc, có thể nói phương thức là một thuộc tính của object có giá trị là một hàm.

```
//Hàm khởi tạo đối tượng
function person(name, age) {
    this.name = name;
    this.age = age;
    this.changeName = function (name) {
        this.name = name;
    }
}

//Tạo đối tượng
var p = new person("Khoa", 19);

p.changeName("Vân");
//Giờ p.name bằng "Vân"
```

Phương thức trong Object

```
function person(name, age) {  
    this.name= name;  
    this.age = age;  
    this.yearOfBirth = bornYear; //Gán phương thức bên ngoài  
}  
  
//Hàm bên ngoài hàm tạo, hàm này được gán vào đối tượng qua hàm tạo ở trên  
function bornYear() {  
    return 2020 - this.age;  
}  
  
var p = new person("Khoa", 19);  
document.write(p.yearOfBirth());  
// Outputs 2001
```

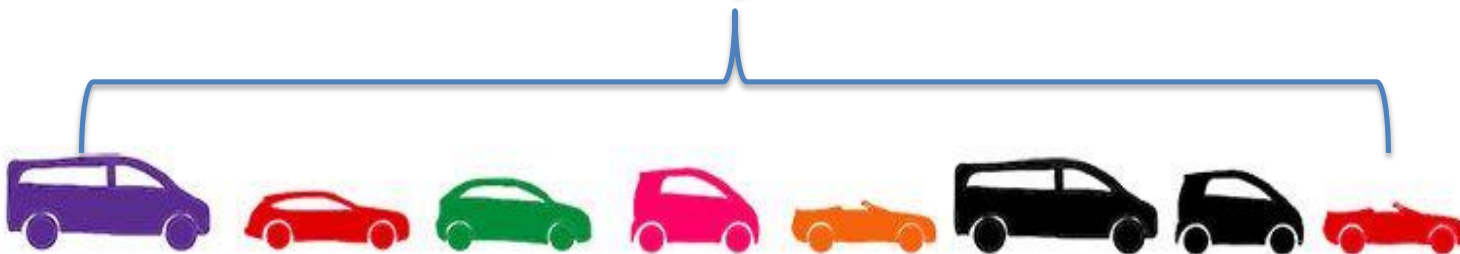
Array of Objects

```
const car = {  
  "color": "purple",  
  "type": "minivan",  
  "registration": new Date('2012-02-03'),  
  "capacity": 7  
}
```

Object: Car



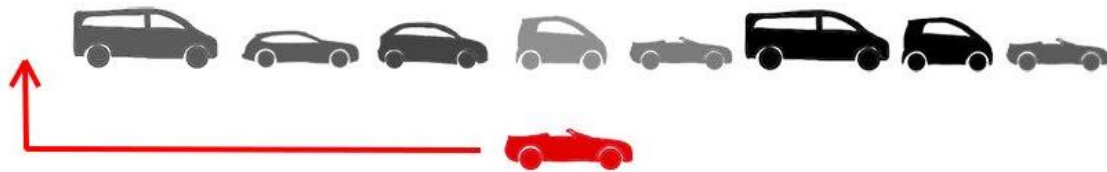
List Of
Objects Car



```
let cars = [  
  {  
    "color": "purple",  
    "type": "minivan",  
    "registration": new Date('2017-01-03'),  
    "capacity": 7  
  },  
  {  
    "color": "red",  
    "type": "station wagon",  
    "registration": new Date('2018-03-03'),  
    "capacity": 5  
  },  
  ...  
],  
...
```

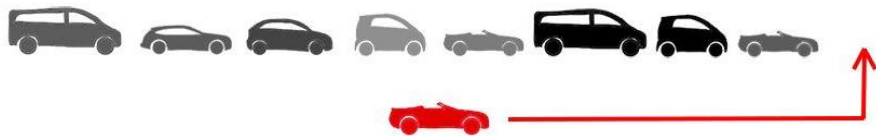
Add a new object

- at the **start** - **Array.unshift**



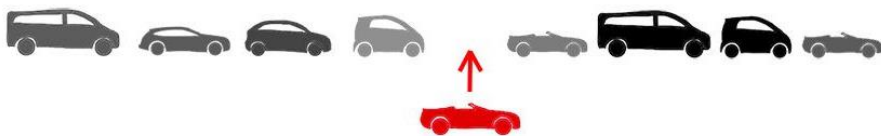
```
let car = {  
  "color": "red",  
  "type": "cabrio",  
  "registration": new Date('2016-05-02'),  
  "capacity": 2  
}  
cars.unshift(car);
```

- at the **end** - **Array.push**



```
let car = {  
  "color": "red",  
  "type": "cabrio",  
  "registration": new Date('2016-05-02'),  
  "capacity": 2  
}  
cars.push(car);
```

- in the **middle** - **Array.splice**



```
let car = {  
  "color": "red",  
  "type": "cabrio",  
  "registration": new Date('2016-05-02'),  
  "capacity": 2  
}  
cars.splice(4, 0, car);
```

```
Array.splice(  
  {index where to start},  
  {how many items to remove},  
  {items to add}  
);
```

Find an object in an array

- Find an object in an array by its values - **Array.find**

🔍 color: red



car = 

```
let car = cars.find(car => car.color === "red");  
console.log(car);
```


Array - Filter

- Get multiple items from an array that match a condition

Q color: red



cars = [red car, red car]

```
let redCars = cars.filter(car => car.color === "red");
console.log(redCars);
// output:
// [
//   {
//     color: 'red',
//     type: 'station wagon',
//     registration: 'Sat-Mar-03-2018-01:00:00-GMT+0100-(GMT+01:00)',
//     capacity: 5
//   },
//   {
//     color: 'red',
//     type: 'cabrio',
//     registration: 'Sat-Mar-03-2012-01:00:00-GMT+0100-(GMT+01:00)',
//     capacity: 2
//   }
// ]
```

- Transform an array of objects into an array of different objects - **Array.map**

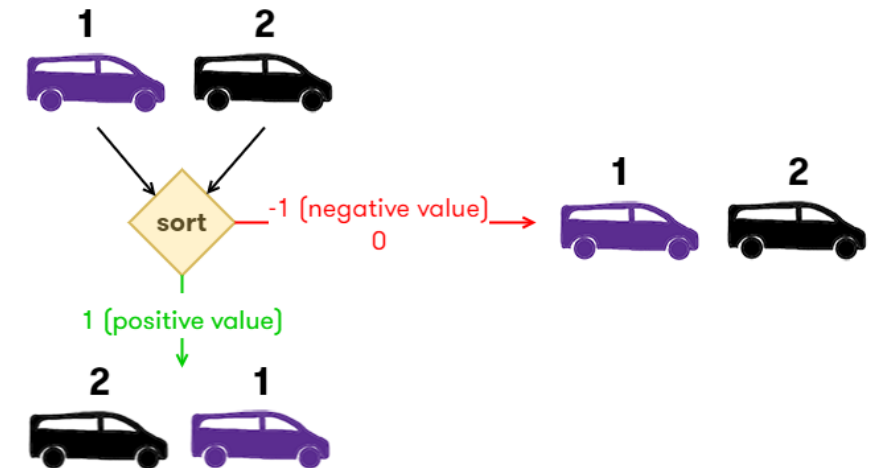
small car = 1-3 people
medium car = 4-5 people
large car = 6-9 people



```
let sizes = cars.map(car => {  
  if (car.capacity <= 3){  
    return "small";  
  }  
  if (car.capacity <= 5){  
    return "medium";  
  }  
  return "large";  
});  
console.log(sizes);  
// output:  
// ['large','medium','medium', ..., 'small']
```

Array - Sort

- Sort an array by a property - **Array.sort**



```
let sortedCars = cars.sort((c1, c2) => (c1.capacity < c2.capacity) ? 1 : (c1.capacity > c2.capacity) ? -1 : 0);  
console.log(sortedCars);
```

Array - Reduce

- The **reduce()** method reduces an array of values down to just one value.
- To get the output value, it runs a reducer function on each element of the array
- **Syntax: arr.reduce(callback[, initialValue])**
 - ✓ Callback
 - ❖ *accumulator* - the returned value of the previous iteration
 - ❖ *currentValue* - the current item in the array
 - ❖ *index* - the index of the current item
 - ❖ *array* - the original array on which reduce was called
 - ✓ initialValue argument is optional.

```
const objs = [  
  {name: "Peter", age: 35},  
  {name: "John", age: 27},  
  {name: "Jake", age: 28}  
];  
  
objs.reduce(function(accumulator, currentValue) {  
  return accumulator + currentValue.age;  
}, 0); // 35 + 27 + 28 = 90
```

Thank you

