**Javascript căn bản**

**Họ và tên: Nguyễn Quốc Tĩnh**

**1. Biến và hằng (Variable)**

1.1. Khai báo

* 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 đấu $;
  + Biến đặt phái 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;

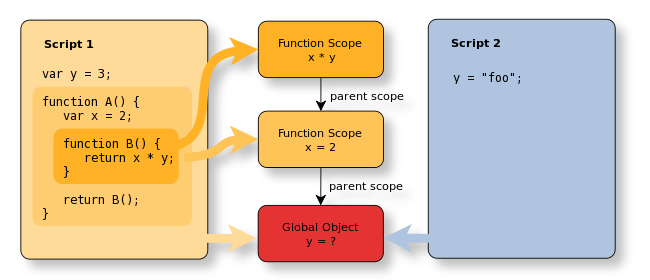
var score = 0, highScore = 0, player = '';

var x, y, z;

var \_money = 0;

1.2. Phạm vi (scope)

* 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*



**2. Kiểu dữ liệu (Data types)**

2.1. Nguyên thủy: Primitive types:

* Khi khái 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 (Undefine) 🡪 Biến được khai báo nhưng không có giá trị
  + Đối tượng (Object)
  + Mảng (Array)
  + Ngày tháng (Date)

<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 Undefine

        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>

2.2. Tham chiếu: Non-primitive types (reference)

* Trong javascript có 3 kiểu dữ liệu tham chiếu "Reference", gọi chung là kiểu dữ liệu Object type, vì nó không mang giá trị mà chỉ tham chiếu đến vùng lưu trữ của object đó trong bộ nhớ, Reference sẽ không lưu giá trị được gán mà nó sẽ chỉ lưu địa chỉ đến đến giá trị của bộ nhớ.:
  + Function
  + Array
  + Object

**3. Toán tử (Operators)**

3.1. 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 |
| = | x = y |
| += | x += y |
| -= | x -= y |
| \*= | x \*= y |
| /= | x /= y |
| %= | x %= y |
| \*\*= | x \*\*= y |

3.2. Các toán hạng thường dùng trong việc so sách 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;** |
| && | Toán hạng Logic và |
| || | Toán hạng Logic hoặc |
| ! | Toán hạng Logic phủ định (not) |

3.3. Độ ưu tiên của các toán tử

|  |  |  |
| --- | --- | --- |
| Loại toán tử | Toán tử | Tính kết hợp |
| Một ngôi | -, ++, -- | Phải sang trái |
| Hai ngôi | ^ | Trái sang phải |
| \*, /, % |
| +, - |
| = | Phải sang trái |

**4. Câu lệnh (Statements)**

**4.1. Phân nhánh**

**4.1.1. Cấu trúc điều kiện If-else**

|  |  |
| --- | --- |
| if (condition) {      //  làm gì đó khi condition là True  } else {      //  Làm gì đó khi condition là False  } | Decision Making |

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

if (condition1) {

    //Làm gì đó khi biểu thức condition 1 là True

} else if (condition2) {

    //Làm gì đó khi biểu thức condition 2 là True

} else if (condition3) {

    //Làm gì đó khi biểu thức condition 3 là True

} else if (condition4) {

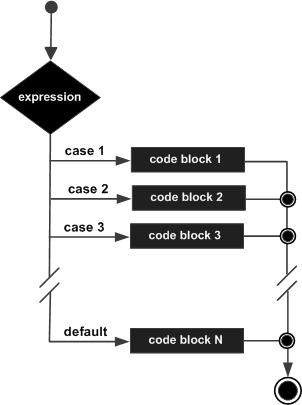
    //Làm gì đó khi biểu thức condition 4 là True

} else {

    //Làm gì đó khi không có biểu thức condition nào ở trên là True

}

**4.1.3.** Muốn kiểm tra giá trị của một biến/Biểu thức nào đó có thể xẩy 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 xẩy ra với expression
* Từ khóa break 🡪 Kết thúc việc thưc 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

**4.2. Vòng lặp**

**4.2.1. Cấu trúc vòng lặp for**

Cú pháp:

|  |  |
| --- | --- |
| for (statement 1; statement 2; statement 3) {       // code block to be executed  } | For Loop |

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

**4.2.2. 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:

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

**4.2.3. 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, …

Cú pháp:

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*

**4.2.4. Vòng lặp While và do-while**

* Vòng lặp while sẽ thực hiện các lệnh bên trong vòng lặp trong khi điều khiện vòng lặp (condition) vẫn còn true

Cú pháp:

|  |  |
| --- | --- |
| while (condition) {      // code block to be executed  } | While loop |
|  |  |

* 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)

Cú pháp:

|  |  |
| --- | --- |
| do {      // code block to be executed  }while (condition); | Do While Loop |

**5. Hàm (Function)**

- Mục đích:

* 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

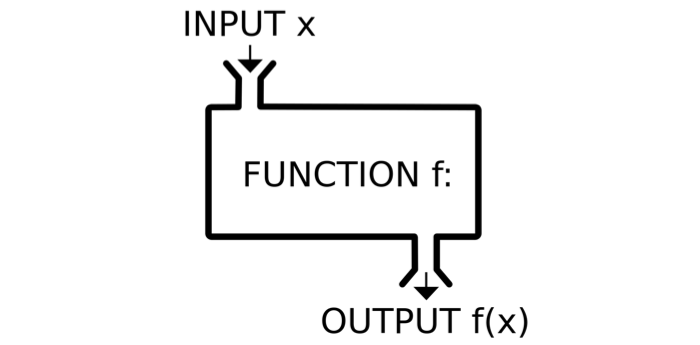
**5.1. Normal function**

* Cú pháp:

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 nghìa 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;
  + paramater1, paramater2, paramater3 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
* 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));

}

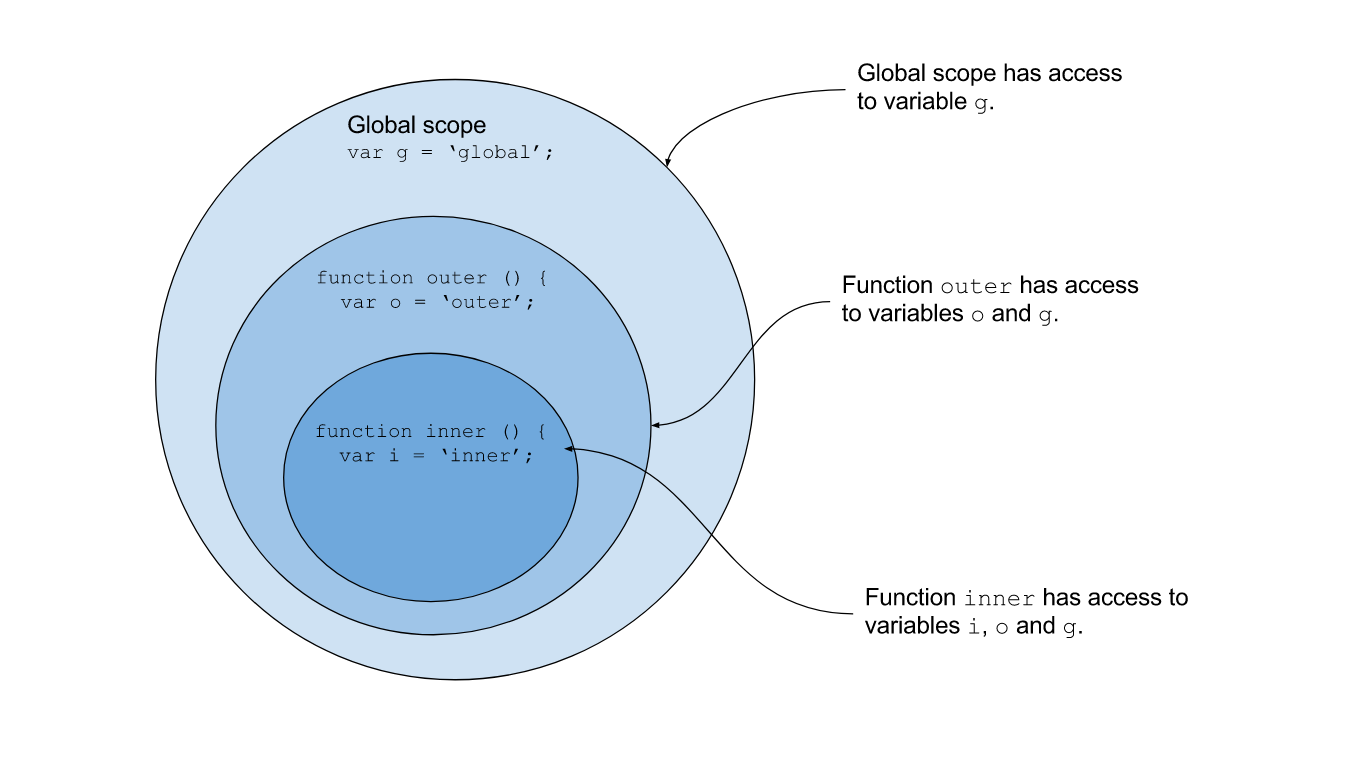
* Hàm anonymous – Hàm không cần đặt tên

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
* *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
  + 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);

**5.2. Arrow function**

Arrow function còn được gọi là hàm mũi tên, tức là ta sẽ tạo hàm bằng cách sử dụng ký tự =>. Cách tạo này xuất hiện kể từ phiên bản ES6.

* Cú pháp:

var name\_of\_function = (var1, var2, var3, ...) => {

    // Nội dung function

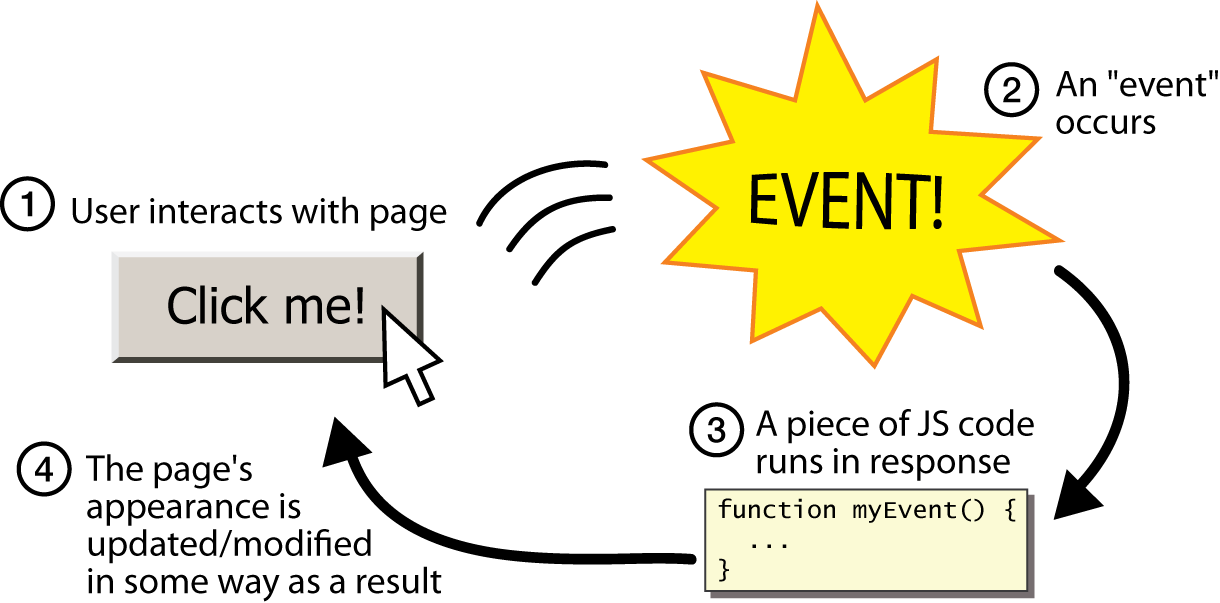
};

**5.3. Callback**

* Callback function có thể được hiểu nôm na như sau: callback tức là ta truyền một đoạn code (Hàm A) này vào một đoạn code khác (Hàm B). Tới một thời điểm nào đó, Hàm A sẽ được hàm B gọi lại (callback). Javascript là một ngôn ngữ lập trình hướng sự kiện và bất đồng bộ nên callback function đóng vai trò rất quan trọng, bạn sẽ truyền một callback function vào các sự kiện và xử lý bất đồng bộ đó..
* Nguyên tắc khi thực hiện Callback Function:
  + Tham số truyền vào phải là một function
  + Cẩn thận với this khi hàm callback nằm trong object
  + Khắc phục this khi hàm callback nằm trong object
  + Callback Hell

**6. Sự kiện (Event)**

* Sự kiện là “**những gì**” được thực hiện hoặc có hành động nào đó xẩy ra trên trình duyệt;
* Một số sự kiện trên trang:
  + Trang web được tải xong
  + Nhấn vào một Nút nào đó, một phần tử nào đó
  + Các phần tử form thay đổi giá trị
  + ….



* **Các cách lắng nghe sự kiện trong JS**
  + **Lắng nghe trong phần tử HTML (Inline/HTML Event handlers)**
  + **Lắng nghe sự kiện sử dụng hàm addEventListener()**
  + **Bỏ lắng nghe sự kiện với hàm removeEventListeners()**

**6.1. Inline/HTML Event handlers**

* Sự kiện được kích hoạt thông qua liên kết giữa phần tử HTML bằng tên thuộc tính của nó
* Tên sự kiện bắt đầu bằng tiền tố on
* Nhưng không phải tất cả các phần tử đều có thể gọi các kiểu sự kiện này được:
  + onchange: Một phần tử HTML đã được thay đổi
  + onclick: Người dùng nhấp vào một phần tử HTML
  + onmouseover: Người dùng di chuột qua phần tử HTML
  + onmouseout: Người dùng di chuyển chuột ra khỏi phần tử HTML
  + onkeydown: Người dùng nhấn phím trên bàn phím
  + onload: Trình duyệt đã tải xong trang
* Ví dụ:
  + HTLM:

<a id="alink" href="#" onclick="doSomething(this);"> Nhấp vào tôi</a>

    <a id="alink" href="javascript:doSomething(this)">Nhấp vào tôi</a>

    <div onmouseover="over()" onmouseout="out()">

        <h2>This is inside the division</h2>

    </div>

* + Javascrip:

<script>

        function doSomething() {

            alert("Hello, You're calling me ...");

        }

        function over() {

            document.write("Mouse Over");

        }

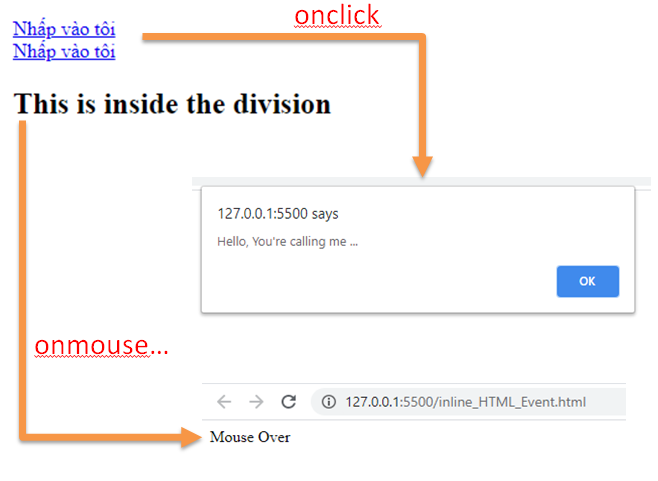
        function out() {

            document.write("Mouse Out");

        }

    </script>

* + Kết quả



**6.2. addEventListener() and removeEventListener()**

* Thêm sự kiện cho phần tử thông qua addEventListene() element.addEventListener(“event”, functionName [,Boolean]);
* Xóa sự kiện của phần tử element.removeEventListener(“event”, functionName [,Boolean]);
* Ví dụ:
  + Code:

<button>Click me</button>

    <p></p>

    <script type="text/javascript">

        var btn = document.querySelector("button");

        function changeColor() {

            btn.style.backgroundColor = "blue"; //change background color

            btn.style.color = "white"; //change font color

            document.querySelector("p").innerHTML = "Great! The button changed its color." //add text

        }

        btn.addEventListener("click", changeColor); //adds event listener

     // btn.removeEventListener(“click, changeColor”); //To remove the event listener

    </script>

* + Kết quả:



**6.3. Các kiểu Sự kiện trong JS**

6.3.1. User Interface events

* Sự kiện này xẩy ra khi có bất kỳ sự tương tác nào lên cửa sổ trình duyệt hơn là lên trang HTML;
* Sự kiện này chúng ta thường lắng nghe trên đối tượng Window, không phải trên đối tượng tài liệu (DOM);
* Một số sự kiện phổ biến:
  + Load 🡪 Đang tải trang Web
  + Unload 🡪 Trang Web chưa tải xong
  + Error 🡪 Có lỗi JavaScript xẩy ra trên trang
  + Resize 🡪 Resize lại kích thước cửa sổ trình duyệt
  + Scroll 🡪 Khi người dùng kéo thanh trượt lên/xuống trên cửa sổ trình duyệt.

6.3.2. Focus and blur events

* Sự kiện này sẽ được kích hoạt khi người dùng focus vào phần tử HTML;
* Loại sự kiện này thường hữu ích khi sử dụng trên các Form với các công việc:
  + Hiển thị tips hoặc feedback cho người dùng khi người dùng tương tác vào phần tử Form;
  + Kiểm tra tính hợp lệ của dữ liệu trên form khi người dùng di chuyển ra khỏi một phần tử HTML mà chưa cần nhấn nút Submit;
* Một số hàm sự kiện:
  + Focus 🡪 xẩy ra trên một nút DOM nào đó mà người dùng đang “ở trong” phần tử đó (focus)
  + Blur 🡪 xẩy ra trên một nút DOM khi người dùng không còn focus vào phần tử
  + Focusin 🡪 tương tự như sự kiện focus. Nhưng Firefox không hỗ trợ sự kiện focusin

Focusout 🡪 tương tự như sự kiện blur. Nhưng Firefox không hỗ trợ sự kiện này

6.3.3.Mouse Event

* onclick — Sự kiện xảy ra khi người dùng nhấp vào một phần tử
* oncontextmenu — Người dùng nhấp chuột phải vào một phần tử để mở menu ngữ cảnh
* ondblclick — Người dùng nhấp đúp vào một phần tử
* onmousedown — Người dùng nhấn nút chuột qua một phần tử
* onmouseenter — Con trỏ di chuyển vào một phần tử
* onmouseleave — Con trỏ di chuyển ra khỏi một phần tử
* onmousemove — Con trỏ đang di chuyển khi nó ở trên một phần tử
* onmouseover — Khi con trỏ được di chuyển vào một phần tử hoặc một trong các phần tử con của nó
* onmouseout — Người dùng di chuyển con trỏ chuột ra khỏi một phần tử hoặc một trong các phần tử con của nó
* onmouseup — Người dùng thả nút chuột khi ở trên một phần tử

6.3.4. Window/Frame Event

* onabort — Quá trình tải phương tiện bị hủy bỏ
* onbeforeunload — Sự kiện xảy ra trước khi tài liệu sắp được dỡ xuống
* onerror — Xảy ra lỗi khi tải tệp bên ngoài
* onhashchange — Đã có những thay đổi đối với phần neo của URL
* onload — Khi một đối tượng đã được tải
* onpagehide — Người dùng điều hướng khỏi một trang web
* onpageshow — Khi người dùng điều hướng đến một trang web
* onresize — Chế độ xem tài liệu được thay đổi kích thước
* onscroll — Thanh cuộn của một phần tử đang được cuộn
* onunload — Sự kiện xảy ra khi một trang đã được tải xuống

6.3.5. Form Event

* onblur — Khi một phần tử mất tiêu điểm
* onchange — Nội dung của một phần tử biểu mẫu thay đổi (<input>, <select> , <textarea>)
* onfocus — Một phần tử được lấy tiêu điểm
* onfocusin — Khi một phần tử sắp lấy tiêu điểm
* onfocusout — Phần tử sắp mất tiêu điểm
* oninput — Người dùng nhập vào một phần tử
* oninvalid — Một phần tử không hợp lệ
* onreset — Một biểu mẫu được đặt lại
* onsearch — Người dùng viết một cái gì đó trong trường tìm kiếm (<input="search">)
* onselect — Người dùng chọn một số văn bản ( <input>, <textarea>)
* onsubmit — Một biểu mẫu được gửi

6.3.6. Drag/Drop Event

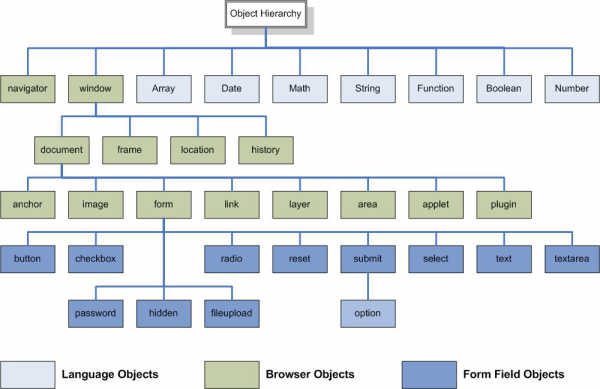
* ondrag — Một phần tử được kéo
* ondragend — Người dùng đã hoàn tất việc kéo phần tử
* ondragenter — Phần tử được kéo đi vào mục tiêu thả
* ondragleave — Một phần tử được kéo rời khỏi mục tiêu thả
* ondragover — Phần tử được kéo nằm trên đầu mục tiêu thả
* ondragstart — Người dùng bắt đầu kéo một phần tử
* ondrop — Phần tử được kéo bị bỏ trên bảng thả

6.3.7. Media Event

* onabort — Tải phương tiện bị bỏ dở
* oncanplay — Trình duyệt có thể bắt đầu phát phương tiện (ví dụ: một tệp đã đủ bộ đệm)
* oncanplaythrough — Trình duyệt có thể phát qua phương tiện mà không cần dừng lại
* ondurationchange — Thời lượng của phương tiện thay đổi
* onended — Các phương tiện truyền thông đã kết thúc
* onerror — Xảy ra khi có lỗi xảy ra khi tải tệp bên ngoài
* onloadeddata — Dữ liệu phương tiện được tải
* onloadedmetadata — Siêu dữ liệu (như thứ nguyên và thời lượng) được tải
* onloadstart —  Trình duyệt bắt đầu tìm kiếm phương tiện được chỉ định
* onpause — Phương tiện bị người dùng tạm dừng hoặc tự động
* onplay — Phương tiện đã được khởi động hoặc không còn bị tạm dừng
* onplaying — Phương tiện đang phát sau khi bị tạm dừng hoặc dừng để lưu vào bộ đệm
* onprogress — Trình duyệt đang trong quá trình tải xuống phương tiện
* onratechange — Tốc độ phát của phương tiện thay đổi
* onseeked — Người dùng đã hoàn tất việc di chuyển / chuyển sang vị trí mới trong phương tiện
* onseeking — Người dùng bắt đầu di chuyển / bỏ qua
* onstalled — Trình duyệt đang cố gắng tải phương tiện nhưng nó không khả dụng
* onsuspend — Trình duyệt cố tình không tải phương tiện
* ontimeupdate — Vị trí chơi đã thay đổi (ví dụ: vì tua đi nhanh)
* onvolumechange — Âm lượng phương tiện đã thay đổi (bao gồm cả tắt tiếng)
* onwaiting — Phương tiện bị tạm dừng nhưng dự kiến ​​sẽ tiếp tục (ví dụ: lưu vào bộ đệm)

**7. DOM**

* DOM = (Document Object Model
* DOM – Mô hình Đối tượng Tài liệu
* Khi trang Web được tải lên, trình duyệt sẽ tạo ra một mô hình đối tượng tài liệu của trang.
* Mô hình DOM HTML được xây dựng như một cây của đối tượng gọi là DOM Tree

****

* DOM là một API cho JavaScript:
  + Có thể thêm/thay đổi/xóa các phần tử HTML
  + Có thể thêm/thay đổi/xóa thuộc tính HTML
  + Có thể thêm/thay đổi/xóa định dạng CSS
  + Có thể phản ứng với các sự kiện HTML
  + Có thể thêm/thay đổi/xóa sự kiện HTML
* Các cách để tìm phần tử trong HTML
  + Tìm phần tử HTML theo id

document.getElementById(id);

* + Tìm các phần tử HTML theo tên thẻ

document.getElementsByTagName(tagname);

* + Tìm các phần tử HTML theo tên lớp

document.getElementsByClassName(classname);

* + Tìm các phần tử HTML bằng bộ chọn CSS

document.querySelectorAll(htmlselector);

* + Tìm phần tử HTML thông qua tập hợp các đối tượng HTML:

anchors, forms**,** images, links và scripts

* Thay đổi phần tử HTML

|  |  |
| --- | --- |
| Đối tượng | Mô tả |
| *element*.innerHTML = “giá trị mới cho phần tử” | Thay đổi giá trị của phân tử HTML |
| *element*.*attribute = “giá trị thuộc tính mới”* | Thay đổi giá trị thuộc tính của phần tử HTML |
| *element*.style.*property = “Thuộc tính CSS mới”* | Thay đổi giá trị CSS của một phần tử HTML |
| Phương thức | Mô tả |
| *element*.setAttribute*(attribute, value)* | Thay đổi giá trị thuộc tính của phần tử HTML |

* Thêm, thay thế và xóa các phần tử

|  |  |
| --- | --- |
| Phương thức | Mô tả |
| document.createElement(*element*) | Tạo mới một phần tử |
| document.removeChild(*element*) | Xóa một phần tử |
| document.appendChild(*element*) | Thêm một phần tử con |
| document.replaceChild(*new, old*) | Thay thể một phần tử |
| document.write(*text*) | Xuất nội dung ra tài liệu HTML |

**8. Các hàm sẵn có trong javascript**

8.1. Number, String

* Các hàm xử lý chuỗi (String) trong JS

|  |  |
| --- | --- |
| Tên hàm | Mô tả |
| 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 |

* Các hàm xử lý số (Number) trong JS

|  |  |
| --- | --- |
| Tên hàm | Mô tả |
| Number(numOjb) | Convert một đối tượng numOjb chứa số về giá trị số**chứa số về giá trị s** |
| 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 |

8.2. 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
* Khai báo mảng trong Js
  + Cách 1: Khai báo cấu trúc mảng truyền thồng:

var array\_name = [item1, item2, ...];

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

var array\_name = new Array(item1,item2,...);

* Các hàm/Phương thức xử lý dữ liệu trong mảng

|  |  |
| --- | --- |
| Tên hàm | Chức năng |
| pop | Lấy/xóa một phân tử ở cuối mảng |
| push(item1, item2, …) | Thêm các phần tử (item) vào cuối mảng |
| toString() | Convert mảng về thành một chuỗi, các giá trị trong mảng sẽ được phân cách nhau bằng dấu phẩy |
| join(separator) | Nối tất cả các phần tử trong mảng thành một chuỗi được phân biệt với nhau bằng một ký hiệu nào đó (separator) |
| splice(start, deleteCount, item 1, item 2….) | 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 |
| sort() | Sắp sếp các phần tử trong mảng |
| shift() | Lấy/xóa một phân tử ở đầu mảng |
| unshift(item1, item2, …) | Thêm phần tử (item) vào đầu mảng |
| find(logic) | Trả về giá trị đầu tiên tìm thấy (thỏa mãn biểu thức điều kiện) |
| concat(otherArray) | Nối otherArray vào array đang có |

8.3. Date time, Math…

* 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 |
| getFullYear() | Trả về năm có 4 số (yyyy) |
| getMonth() | Trả về tháng (0-11) |
| getDate() | Trả về ngày (1-31) |
| getHours() | Trả về giờ (0-23) |
| getMinutes() | Trả về phút (0-59) |
| getDay() | Trả về ngày trong tuần dưới dạng số (0-6), 0 🡪 Chủ nhật |
| setFullYear() | Thiết lập lại giá trị năm (4 số) |
| setMonth() | Thiết lập lại giá trị tháng (0-11) |
| setDate() | Thiết lập lại giá trị ngày (1-31) |
| setHours() | Thiết lập lại giá trị giờ (0-23) |
| setMinutes() | Thiết lập lại giá trị phút (0-59) |
| setDay() | Thiết lập lại giá trị ngày trong tuần (0-6) |

* Các hàm toán học (Math)

|  |  |
| --- | --- |
| Tên hàm | Mô tả |
| round(num) | Trả về giá trị num đã được làm tròn (Về giá trị số nguyên gần nó nhất) |
| pow(x,y) | Trả về giá trị của x mũ y |
| sqrt(x) | Trả về giá trị căn bậc hai của x |
| abs(x) | Trả về giá trị tuyệt đối của x |
| ceil(x) | Trả về giá trị x được làm tròn lên số nguyên gần nó nhất |
| floor(x) | Trả về giá trị x được làm tròn xuống số nguyên gần nó nhất |
| min(list\_num) | Trả về giá trị nhỏ nhất trong danh sách các số (list\_num) |
| max(list\_num) | Trả về giá trị lớn nhất trong danh sách các số (list\_num) |
| random() | Trả về một gia trị số thực ngẫu nhiên nằm trong khoảng [0-1] |