|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **JAVASCRIPT TOTURIAL** | | | | |
| **Variable** | | **Biến** | | |
| Có 3 cách để khai báo một biến trong JavaScript   * **var** // ***có thể redeclare*** , khai báo sau khi sử dụng được   var x = 2;  var x = 3; // allowed | x = 3; //allowed   * **let** // block scope , ***không thể redeclare***, phải khai báo trước khi sử dụng, **có thể gán sau**   let x = 2;  let x = 3; //not allowed | x = 3; //allowed   * **const** // block scope , ***không thể redeclare***, phải khai báo trước khi sử dụng, **không thể gán sau**   const x = 2;  const x = 3; //not allowed | x = 3; //not allowed | | | | |
| **Functions** | | **Hàm** | | |
| **Có 3 kiểu khai báo hàm:**   * **function** *myFunction***(** p1,p2 **) {**   **return** p1\*p2**;**  **}**   * **var** *myFunction* **= new Function(“** p1 **”,”** p2 **”,” return** p1\*p2; **“);** * **var** *myFunction* **= function(**p1,p2**){ return** p1\*p2 **}**;   **Để gọi hàm ta sử dụng toàn tử call ( ) :**  **vd: myFunction( );**  **Đối tượng Arguments trong hàm:**  > arguments là một đói tượng có sẵn trong Function, nó là một mảng chứa tập hợp các tham số truyền vào <  Có thể dùng **for.. of** để lặp qua các giá trị của **arguments**  vd: function *myFunction*() {  let text = “”;  **for**( let x **of** **arguments**){  text += x;  }  return text;  }  let myText = *myFunction*( “a”, 1, “b”, “C” ); 🡪 myText = “a1cC” | | | | |
| **Object** | | **Đối tượng** | | |
| **- Có 4 cách để tạo một Objects mới:**   * **Tạo một Object duy nhất:**   const person = **{**  firstName**:** “*Binh*”, //property  [“lastName”]**:** “*Tran*” , //property  [“  fullName**:** function(){ return this.firstName + “ “ + this.lastName, } //method  **}**;  \*\* **có thể thêm property** cho object như sau: *person.haircolor = ”brow” ;*  \*\* **cũng có thể xoá property** cho object như sau: ***delete*** *person.haircolor;*  *\*\** **có thể thêm method vào Object như sau:** person.name = **function()** { return this.firstName + “ “ +  this.lastName;};   * **Tạo một đối tượng với từ khoá NEW (new Object() ) :**   const person = **new Object( );**  person.firstName **=** “*Binh*”, //property  person.lastName **=** “*Tran*”, //property   * **Xác định phương thức tạo đối tượng, sau đó tạo các Objects bằng Object Constructor**   **function** person(firstName,lastName)**{** //**tạo Object Constructor**  this.firstName = firstName;  this.lastName = lastName;  this.fullName = function() { return this.firstName +” “+this.lastName;};  **}**  const student1= **new** person(“Binh”,”Tran”); //**tạo Object** từ **Object Constructor**    **\*\*** **để thêm thuộc tính hoặc method vào Object Constructor ta dùng:** **prototype**  **\*\* để xem Object Constructor của một Object obj ta sử dụng : console.log(obj.constructor);**   * **Tạo một đối tượng bằng sử dụng Object.creat()**   **- Truy cập vào thuộc tính và phương thức của đối tượng:**  Có 2 cách truy cập thuộc tính:  **objectName.propertyName** 🡪 person.firstName; //=”Binh”  **objectName.[“propertyName”]** 🡪 person.[“firstName”] //=”Binh”  Truy cập phương thức:  objectName.methodName() 🡪 person.fullName(); //=”Binh Tran”  Nếu không có dấu ( ) nó trả về định nghĩa method //=” function(){ return this.firstName + “ “ + this.lastName; }”  **- Truy cập vào toàn bộ KEYS của đối tượng bằng for…in Loop:**  **text=” “**  **for (**let ***key* in** objectName**) {**  text += objectName[***key***]  **}**  **- Truy cập vào toàn bộ Value của đối tượng bằng for…of Loop:**  **text=” “**  **for (**let **value of Object.values(**objectName**) {**  text += ***value***  **}**  // vì objectName không có thuộc tính lập lại nên ta sử dụng Object Method: **Object.values( )** để tạo thành mảng chứa các giá trị của objectName, từ đó mới có thể dùng **for … of** được.  **Object Methods ES5:**   * **Object.creat( ) : Tạo Object**   Tạo một Object với Object đã có như là một thuộc tính   * **Object.defineProperty( object, property, descriptor ) : Xác định 1 thuộc tính**   Thêm hoặc thay MỘT thuộc tính  Tương tự objectName.newproperty = value,  Khác là với descriptor ta có thể xác định meta data của thuộc tính như :  **writable , emumerable, configurable : true | false;**  **Vd: const person = {**  **firstName: “ Binh ”,**  **lastName : “ Tran ”**  **}**  **Object.defineProperty( person, “language”, { value: “NO”, writable: false} )**  **\*\*\* Với Object.defineProperty ta có thể thêm getter và setter để tạo các phương thức lấy hoặc thay đổi giá trị thuộc tính trong Object;**  **vd: const person = {**  **firstName: “ Binh ”,**  **lastName : “ Tran ”**  **Object.defineProperty( person , getFullName, {**  **get: function() { return this.firstName + this.lastName}**  **});**  **Object.defineProperty( person , setFirstName, {**  **set: function( newValue ) { this.firstName = newValue**  **});**  **person.setFirstName(“ Ngoc Bao Binh”);**  **console.log( person.getFullName() ); // “ Tran**   * **Object.defineProperties( object, descriptors ) : Xác định nhiều thuộc tính**   Thêm hoặc thay NHIỀU thuộc tính  vd: **const person = { }**  **Object.defineProperties( person, {**  **firstName: {**  **value: “ Binh ”,**  **writable: false**  **},**  **lastName {**  **value: “ Binh ”,**  **writable: false**  **},**  **} );**  **// Lúc này person = { firstName: “ Binh ”, lastName : “ Tran ” }**  **\*\*\* Với Object.defineProperty ta có thể thêm getter và setter để tạo các phương thức lấy hoặc thay đổi giá trị thuộc tính trong Object;**  **vd: const person = {**  **firstName: “ Binh ”,**  **lastName : “ Tran ”**  **}**  **Object.defineProperties( person , {**  **getFullName : {**  **get: function() { return this.firstName + this.lastName }**  **},**  **setFirstName: {**  **set: function( newValue ) { this.firstName = newValue }**  **} );**  **person.setFirstName(“ Ngoc Bao Binh”);**  **console.log( person.getFullName() ); // “ Tran**   * **Object.getOwnPropertyNames( object ) : Nhận mô tả các tên thuộc tính**   **Trả về các tên properties dưới dạng mảng**   * **Object.getPrototypeOf( object ) : Truy cập vào nguyên mẫu** * **Object.keys( object ) : Trả về các Propertise có thể đếm được dưới dạng mảng** * **Object.preventExtensions( object ) : Ngăn không cho thêm thuộc tính vào object** * **Object.isExtensible( object ) : Trả về TRUE nếu đối tượng đang ngăn không cho thêm thuộc tính** * **Object.seal( object ) : Ngăn không cho thay đổi thuộc tính ( không phải giá trị thuộc tính )** * **Object.isSealed( object ) : Trả về TRUE nếu object đang được seal** * **Object.freeze( object ): Ngăn chặn mọi thay đổi đối với object** * **Object.isFrozen( object ): Trả về TRUE nếu object đang đóng băng**   **get: function( ) { return this.*propertyName* }**  **set: function( *newValue* ) { this.*propertyName* = *newValue* }** | | | | |
| **Strings** | | **Chuỗi** | | |
| **Lưu ý khi sử dụng String:**   * **Muốn bỏ dấu “ “ vào trong chuỗi**   let text = **‘** Đây là đoạn văn có **“**dấu ngoặc kép**”** bên trong **‘** ; -- dùng “ ” đặt bên trong ‘ ‘  hoặc: = **“** Đây là đoạn văn có **\“**dấu ngoặc kép**\”** bên trong **“** ; -- dùng \ trước dấu “   * **Khi đoạn text dài hơn 80 dòng ta có 3 cách để làm ngắn lại:**   - Xuống dòng sau toán tử: document.getElementById(‘text’).innerHTML **=**  ‘ Đây là một đoạn văn dài ơi là dài ‘;  - Dùng dấu \ trong text string: document.getElementById(‘text’).innerHTML **= “** Đây là **\**  một đoạn văn dài ơi là dài “;  - Dùng dấu + chia đôi text: document.getElementById(‘text’).innerHTML **= “** Đây là” +  “ một đoạn văn dài ơi là dài “;  - Hoặc sử dụng dấu **Back – Tics ` `** : document.getElementById(‘text’).innerHTML **=** `Đây là  một đoạn văn dài ơi là dài **`** ;  **String Methods:**  **let text = ‘tran ngoc bao binh’;**   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | **Độ dài** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |  | | **string** | **“t** | **r** | **a** | **n** |  | **n** | **g** | **o** | **c** |  | **b** | **a** | **o** |  | **b** | **i** | **n** | **h** | **”** | | **vị trí +** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | | **vị trí -** | -18 | -17 | -16 | -15 | -14 | -13 | -12 | -11 | -10 | -9 | -8 | -7 | -6 | -5 | -4 | -3 | -2 | -1 | 18 |  * **Length: trả về độ dài của string**   text.leng = 18   * **slice(start , end ) :**   **Tạo một chuỗi mới từ việc cắt lát chuỗi dựa vào vị trí chuỗi; có thể xài cả vị trí + và vị trí -**   |  |  |  |  | | --- | --- | --- | --- | |  | **n** | **g** | **o** | | 4 | 5 | 6 | 7 | | -14 | -13 | -12 | -11 |   text.slice(4,7); = “ ng”  text.slice(4,-11); = “ ng”  text.slice(-14,-11); = “ ng”   * **substring(start,end)**   **Tạo một chuỗi mới từ việc cắt lát chuỗi dựa vào vị trí chuỗi; chỉ xài vị trí +**  // nếu nhập vào giá trị - sẽ hiểu là từ đầu chuỗi   |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | **Độ dài** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | | **string** | **“t** | **r** | **a** | **n** |  | **n** | **g** | **o** | | **vị trí +** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | | **vị trí -** | -18 | -17 | -16 | -15 | -14 | -13 | -12 | -11 |   text.substring(4,7); = “ ng”  text.substring(-14,4); = “tran”  text.substring(4,-11); = null   * **substr(start, *length)***   **Tạo một chuỗi mới từ việc cắt lát chuỗi dựa vào vị trí chuỗi và độ dài đếm từ vị trí đó; xài vị trí + và –**   |  |  |  |  |  | | --- | --- | --- | --- | --- | | string |  | **n** | **g** | **o** | | vị trí + | 4 | 5 | 6 | 7 | | vị trí - | -14 | -13 | -12 | -11 | | độ dài từ **start** | 1 | 2 | 3 | 4 |   text.substr (4,4); = “ ngo”  text.substr (-14,3); = “ ng”  *nếu length để trống có nghĩa là đến hết chuỗi*  text.substr(-14); = “ ngoc bao binh”   * **replace( )**   **Tạo một chuỗi mới từ việc thay thế một giá trị trong chuỗi bằng một giá trị khác**  **replace(“text cần thay”,”text thay thế”);**  vd: text.replace(“ “,”-“); = tran-ngoc bao binh //chỉ thay thế *text cần thay* tìm được đầu tiên  - Khi sử dụng “text cần thay“ chỉ có thể thay thế được text giống chính xác text cần thay  - Để hiệu quả hơn ta nên dùng RegExp: **replace( RegExp ,”text thay thế”)**  vd: text.replace(/\s/g,”-“); = “tran-ngoc-bao-binh”   * **toUpperCase() và toLowerCase()**   **Tạo chuỗi mới từ việc viết HOA hoặc viết thường toàn chuỗi**  text.toUpperCase() ; = “TRAN NGOC BAO BINH”   * **concat()**   **Tạo chuỗi mới từ việc ghép các chuỗi với nhau**  text.concat(“ “,”hoc javascript”); = “tran ngoc bao binh hoc javascript”  // khá giống: text + “ “ + “hoc javascript”;   * **trim()**   **Tạo chuỗi mới từ việc bỏ white space ở đầu và cuối của string**  let str= “ tran ngoc bao binh ”;  let newStr = str.trim(); = “tran ngoc bao binh”   * **padStart() và padEnd()**   **Tạo chuỗi mới từ việc thêm N ký tự vào đầu hoặc cuối chuỗi**  **padStart(N,”text”)** và **padEnd(N,”text”)** // N số lần muốn lập lại text  padStart(4,”-“); = ----tran ngoc bao binh  padEnd(4,”x”); = tran ngoc bao binhxxx   * **charAt() và charCodeAt()**   **Xuất ra ký tự và mã UTF-16 của ký tự trong chuỗi dựa vào vị trí +**  charAt(vị trí +) , charCodeAt( vị trí +)  text.charAt(5); = “n”  text.charCodeAt(5); = 110   * **[ ]**   **Xuất ra ký tự của String dựa vào vị trí +**  text[5] = “n” , giống với text.split(“”)[5]   * **split()**   **Biến String thành Array dựa vào ký tự để chia**  **split(“text”);**  text.split(“ “); =[‘tran’,’ngoc’,’bao’,’binh’]  **Các Method để tìm kiếm trong String:**   * **indexOf( ) và lastIndexOf( )**  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | t | r | a | n |  | n | g | o | c |  | b | a | o |  | b | i | n | h |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | t | r | a | n |  | n | g | o | c |  | b | a | o |  | b | i | n | h |   **Trả về vị trí của lần xuất hiện đầu tiên hoặc cuối cùng của text cần tìm, có thể cài Position**  text.indexOf(“n”); = 3  text.lastIndexOf(“n”) = 16  ***Nếu không tìm thấy sẽ trả về giá trị là -1:***  text.indexOf(“x”) ; = -1  **Có thể cài đặt vị trí bắt đầu tìm ở indexOf( ) hoặc vị trí cuối cùng tìm ở lastIndexOf( );**   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | ~~t~~ | ~~r~~ | ~~a~~ | ~~n~~ |  | ~~n~~ | g | o | c |  | b | a | o |  | b | i | n | h |   text.indexOf(“n”,6); = 16  text.lastIndextOf(“n”,5) = 5   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | t | r | a | n |  | n | ~~g~~ | ~~o~~ | ~~c~~ |  | ~~b~~ | ~~a~~ | ~~o~~ |  | ~~b~~ | ~~i~~ | ~~n~~ | ~~h~~ |  * **search( )**  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | t | r | a | n |  | n | g | o | c |  | b | a | o |  | b | i | n | h |   **Trả về vị trí xuất hiện đầu tiên của text cần tìm hoặc của RegExp**  text.search(“n”) = 3;   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | t | r | a | n |  | n | g | o | c |  | b | a | o |  | b | i | n | h |   text.search(/\s/) = 4;  ***Nếu không tìm thấy sẽ trả về giá trị là -1***   * **match( )**   **Trả về một mảng các kết quả phù hợp với RegExp**  let text = ”ABC abc 123”;   |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | A | B | C |  | a | b | c |  | 1 | 2 | 3 |   text.match(/ab/ig); = [ ‘AB’ , ‘ab’ ]   * **includes( ) hoặc includes( ‘text’ , start) // vị trí bắt đầu tìm kiếm**   **Trả về giá trị boolean về việc chuỗi có chữa ký tự cần tìm hay không**   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | t | r | a | n |  | n | g | o | c |  | b | a | o |  | b | i | n | h |   text.includes(“ngo”); = true  text.includes(“xx”); = false   * **startsWith( ) và endsWith( ):**   **Trả về giá trị boolean về việc chuỗi có chữa ký tự cần tìm tại vị trí bắt đầu hoặc kết thúc không**  text.startsWith(“tr”); = true  text.endsWith(“nh”); = true  **Có thể cài vị trí bắt đầu trong startsWith(“text”,start):**   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | ~~t~~ | ~~r~~ | ~~a~~ | ~~n~~ |  | ~~n~~ | ~~g~~ | ~~o~~ | ~~c~~ |  | b | a | o |  | b | i | n | h |   text.startsWith(“tr”,10 ); = true  **Có thể cài độ dài kết thúc trong endsWith(“text”,length):**   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | t | r | a | n |  | n | g | o | c |  | ~~b~~ | ~~a~~ | ~~o~~ |  | ~~b~~ | ~~i~~ | ~~n~~ | ~~h~~ |   text.endsWith(“c”,10); = false  **\*\* Javascript String Interpolation : Phép nội suy trong String**  Chỉ dùng được với dấu Back-tics **` ${…..} `**  Dùng để nội suy các biến và biểu thức trực tiếp bên trong chuỗi.  vd: **Variable**  let firstName = “Binh”;  let lastName = “Tran”;  let fullName = **`**Tên đầy đủ là **${**firstName} **${**lastName**}**!**`**; 🡪 Tên đầy đủ là Binh Tran  **Function**  function sum(a,b){ return a+b; }  let tong = **`** Tổng là **${**sum(2,6)**}`**; 🡪 Tổng là 8 | | | | |
| **Number** | | **Số** | | |
| **- Số trong Javascript chỉ có 1 loại, có thể viết có hoặc không có decimals đều được**  let x = 3.14;  let x = 3;  **- Với những số dài thì có thể sử dụng ký hiệu exponent (e = 10)**  let x = 123e5; //=123\*10^5 = 12300000  **- Với các số interger Javascript có thể tính toán chính xác lên đến *15 con số* , nhưng với số float thì không chính xác đâu 😊:**  let x = 0.2 + 0.1; // x = 0.300000000000000004  Để giải quyết ta nên chuyển số dạng float về dạng int rồi sau đó chuyển về lại float:  let x = (0.2\*10 + 0.1\*10)/10;  **- Thêm số và chuỗi:**  let x = “10” + 2 + 3; // x = ‘1023’ -- Tự động chuyển về dạng string từ vị trí ngay trước string và bên trái  let x = 2 + 3 + “10” // x = ‘610’  let x = “10”**/**2; // x = 5 -- Tự động chuyển hết về Number với các toán hạng khác +  **- NaN - Not A Number:**  let x = 10/’two’; // x = NaN;  let x = NaN + 10; // x = NaN; -- Các phép toán với NaN đều cho kết quả là NaN  let x = NaN + “10”; // x = “NaN10”  **- Dùng hàm global Javascript isNaN( ) để xác định một số có phải dạng NaN hay không:**  isNaN(x); // return true hoặc false  **- Infinity hoặc – Infinity: Dương vô cực hoặc Âm vô cực:**  let x = 2/0; // x = Infinity  **\*\* Number Methods: Các phương thức của Numbers**   * **toString( )**   **Trả về dạng String của số**  let x = 123;  x.toString( ); // =‘123’ =x.toString(10) – Hệ thập phân – Mặc định  x.toString(2); // =’1111011’ - Hệ nhị phân   * **toExponential( )**   **Trả về dạng String với một số được làm tròn dựa vào số đầu tiên**  let x = 12345.6543;  x.toExponential(); //=’1.23456543e+4’ --Tất cả số sau số đầu tiên  x.toExponential(2); //=’1.23e+4’ -- 2 số sau số đầu tiên  x.toExponential(4); //=’1.2346e+4’ -- 4 số sau số đầu tiên   * **toFixed( )**   **Trả về dạng String một số được làm tròn dựa vào dấu thập phân**  let x = 12345.6543;  x.toFixed(); //=’12346’ --Làm tròn trước dấu thập phân  x.toFixed(2); //=’12345.65’ --Làm tròn sau dấu thập phân 2 đơn vị   * **toPrecision( )**   **Trả về dạng String một số được làm tròn dựa vào độ dài mong muốn**  let x = 12345.6543;  x.toPrecison(4); //= ‘1.235e+4’ -- Làm tròn sau 4 số đầu tiên   * **valueOf( )**   **Trả về giá trị của biến, không thay đổi gì hết**   * **Converting Variables to Numbers**   **Chuyển giá trị của biến thành số**   * **Number() – Trả về giá trị Number của biến, có thể chuyển đổi Date, Boolean thành Number**   Number(true); //=1  Number(false); //=-1  Number(new Date(“2021-09-03”)); //=1630627200000 , milisecond từ ngày “1970-01-01”  Number(“10 year”); //=NaN , không được chứa space   * **parseFloat() – Trả về giá trị của biến dạng Float**   parseFloat(“10.01”);/ //=10.01  parseFloat(“10 year”); //=10  parseFloat(“year 10”); //=NaN , được chứa whitespace và chỉ chọn số đầu tiên  parseFloat(“10 20”); //=10     * **parseInt() – Trả về giá trị Number của biến dạng Interger**   parseInt(“10.01”); //= 10  parseInt(“10 year”); //= 10  parseInt(“year 10”); //= NaN , được chứa whitespace và chỉ chọn số đầu tiên  parseInt(“10 20”); //= 10  **\*\* Number Properties: các thuộc tính có sẵn của Number**   * **Number.MAX\_VALUE**   let x = Number.MAX\_VALUE; //x = 1.7976931348623157e+308   * **Number.MIN\_VALUE**   let x = Number.MIN\_VALUE; //x = -1.7976931348623157e+308   * **Number.POSITIVE\_INFINITY**   let x = Number.POSITIVE\_INFINITY; // x = + ∞   * **Number.NEGATIVE\_INFINITY**   let x = Number.NEGATIVE\_INFINITY; // x = - ∞   * **Number.NaN**  let x = Number.NaN; //x = NaN | | | | |
| **ARRAY** | | **Mảng** | | |
| **Mảng dùng để lưu nhiều giá trị cho một biến. Cú pháp tạo mảng:**  **const array\_name = [item1, item2, item3,…];**  hoặc xuống dòng tuỳ ý:  **const array\_name = [**  **item1,**  **item2,**  **………**  **];**  hoặc có thể khái báo mảng trước và thêm giá trị sau:  **const array\_name = [ ];**  **array\_name[0] = item1;**  **array\_name[1] = item2;**  **…………..**  hoặc có thể dùng new Array: ( *hạn chế*)  **const array\_name = *new* Array(item1,item2,item3,…);**  **Có thể truy xuất và thay đổi giá trị của Array Element:**  const cars = [“bmw”, “audi”, “toyota”];  Truy xuất element: let x = cars[1]; // x = “audi”  Thay đổi giá trị mảng: car[0]=”mercedes”; //cars=[“mercedes”, “audi”, “toyota”]  **Sự khác nhau giữa Array và Object:**  **Arrays use NUMBERED indexes and Objects use NAMEDS indexes**  -- Mảng dùng chỉ mục được đánh số còn Đối tượng dùng chỉ mục được đặt tên —  **-- Array Key Type: Number & Object Key Type: String and Symbols —**  **Array Properties and Methods :**   * **length - độ dài của mảng**   const fruits = [“cam”, “xoài”, “ổi”, “quýt”]  **fruits.length**; // = 4  **- Truy xuất đến Element cuối cùng của mảng:**  **fruits[fruits.length-1]**; //=”quýt”  **- Lặp qua các giá trị trong mảng – dùng vòng lặp for:**  **for( let i=0; i < fruits.length; i++){ ….. fruists[i] …. }**   * **typeof** fruits**; //= object** * **Array.isArray(**fruits**) ; //= true (dùng nhận biết một mảng)** * fruits **instanceof Array**; **//=true (dùng nhận biết một mảng)**   **\*\* Các Methods làm thay đổi giá trị của Mảng:**   * **.sort( ) – Sắp xếp tăng dần các giá trị theo kiểu Strings (kể cả giá trị là Number):**   Xét ký tự **đầu tiên** ở mỗi value và xếp theo thứ tự: **ký hiệu < chữ số < chữ < chữ có dấu**  *const* fruits = [“cam”, “xoài”, “ổi”, “quýt”,10,35,5,”$”]  **fruits.sort( );** //fruits=[“**$**”, **1**0, **3**5, **5**, “**c**am”, “**q**uýt”, “**x**oài”,”**ổ**i”]  **fruits[0];** //=”$”;   * **.reverse( ) - Ngược lại so với sort**   **TIP:**  **1. Dùng .sort( ) để sắp xếp mảng có giá trị là Number theo kiểu NUMBER:**  - Truyền vào trong sort( ) một hàm để so sánh:  **.sort( function(a,b){return a - b} ) 🡪 Mảng tăng dần**  **.sort( function(a,b){return b - a} ) 🡪 Mảng giảm dần**  - Sau đó có thể dùng để tìm giá trị **MAX** và **MIN** trong mảng bằng **[0]** và **[array.length -1]**  **2. Dùng sort( ) để sắp xếp vị trí ngẫu nhiên:**  **.sort(function(a,b){return 0.5 – Math.random() })**  **3. Dùng sort( ) để đảo ngược thứ tự sắp xếp của mảng:**  **.sort(function(a,b){return -1})**   * **.pop() – xoá phần tử cuối,** trả về string của phần tử đó **& .push(***item***) – Thêm phần tử vào cuối** * **.shift() – xoá phần tử đầu,** trả về string của phần tử đó **& .unshift(***item***) – Thêm phần tử vào đầu** * **[ ] – thay đổi giá trị phần tử tại vị trí xác định**   *const* fruits = [“cam”, “xoài”, “ổi”, “quýt”];  fruits[0]; // xoá value “cam” nhưng để lại một lỗ - undifined trong mảng. – không nên xài   * **.splice( ) – Thêm phần tử vào vị trí chỉ định và xoá bao nhiêu phần tử tại đó (splice : ghép nối)**   **.splice(** vị trí + **,** muốn xoá bao nhiêu phần tử **,** phần tử muốn thêm vào, .. , .. **)**  *const* fruits = [“cam”, “xoài”, “ổi”, “quýt”];  fruits.splice(2, 1, “newItem”); // tại vị trí [2] xoá 1 phần tử và thêm vào “newItem”  🡪 fruits = [“cam”, “xoài”, newItem”, “quýt”];  **\*\* Các Methods không thay đổi giá trị Mảng:**   * **.toString() - Tạo một String là các giá trị trong mảng , cách nhau bằng dấu phấy**   **const fruits=**[“cam”, “xoài”, “ổi”, “quýt”]  fruits.toString(); //=”cam,xoài,ổi,quýt”   * **.join( ) – Tạo một String là các giá trị trong mảng và thêm vào ký tự xen giữa**   **const fruits=**[“cam”, “xoài”, “ổi”, “quýt”]  fruits.join(“**\*”**); //=”cam**\***xoài**\***ổi**\***quýt”   * **.concat( ) – Tạo mảng mới bằng các ghép thêm nhiều mảng hoặc item vào mảng:**   *const* fruits = [“cam”, “xoài”, “ổi”, “quýt”];  const animals =[“chó”,”mèo”];  const allItems = fruits.concat( animals ); // allItems = [“cam”, “xoài”, “ổi”, “quýt”, “chó”,”mèo”];   * **.slice(***start,end***) – Tạo mảng mới bằng cách copy một lát cắt ra ở mảng cũ (slice: một lát,miếng)** * **.forEach( myF*unction* ) – Thực hiện hàm myFunction cho mỗi giá trị của mảng;**   array.forEach(myFunction); giống với for(i=0; i<array.length; i++){ myFunction(array[i]) };   * **.map( myFunction ) - Tạo mảng mới bằng kết quả của việc thực hiện hàm myFunction với**   **mỗi giá trị của mảng:**  newArray = array.map(myFunction); //newArray = [result1,result2,….]   * **.filter( myCheck) - Tạo mảng mới bằng việc kiểm tra tất cả các giá trị của mảng:**   const myArray = [ 12, 16, 28, 29];  newArray = myArray.filter( function(*value, index, itself*){ return *value* > 18} ); //newArray = [28,29]   * **.reduce( myFunction , 0) -**   **Từ trái qua : Truyền vào các cặp giá trị để thực hiện hàm tới khi còn giá trị duy nhất**  **Giá trị khởi tạo ban đầu là 0, giống việc cài ban đầu: let result = 0;**  **Nếu không có giá trị khởi tạo, thì ban đầu accumulator sẽ là giá trị value đầu tiên**  **function myFunction( accumulator, currentValue, currentIndex, originArray){ }**   * **.reduceRight( myFunction ) – Từ phải qua** * **.every( myCheck ) – Trả về TRUE nếu tất cả giá trị của mảng thoả mãn myCheck** * **.some( myCheck ) – trả về TRUE nếu có 1 hoặc vài giá trị thoả mãn myCheck** * **.find( myCheck ) – Trả về GIÁ TRỊ đầu tiên trong mảng thoả mãn myCheck** * **.findIndex( myCheck ) – Trả về VỊ TRÍ + của giá trị đầu tiên thoả mãn myCheck** * **.indexOf( ) và .lastIndexOf( ) – Trả về vị trí + của giá trị cần tìm trong mảng , không có trả về -1** * **.includes( ) hoặc includes( ,start) – trả về TRUE hoặc FALSE nếu mảng có chứa giá trị cần tìm**   >> .includes( ) cho phép kiểm tra giá trị NaN , Khác với .indexOf( )   * **Array.form( ) – Tạo ra mảng từ bất cứ Object nào có thuộc tính length hoặc có thể lặp lại**   const myArray = **Array.from(**“abcdef”**)**; // myArray=[‘a’,’b’,’c’,’d’,’e’,’f’]  // Giống với string.**split(“”);**   * **Array.keys( ) - Trả về một mảng Number các vị trí + trong mảng , gọi là các key** * **.hasOwnProperty(** *key* **) : Một thuộc tính bên bài Object, ở đây dùng kiểm tra key này có phải được chỉ định làm thuộc tính của nó không, hay là được thừa kế.**   **vd:** myArray.hasOwnProperty(‘toString’); // false – vì thuộc tính toString được thừa kế từ Array Object  **\*\* Array Const:**  **Cannot be Reassigned:**  const cars = [ “toyota”, “audi”, “bmw”];  car=[“merc”, “volvo”, “toyota”]; //ERROR  **But Elements Can Be Reassagined:**  const cars = [ “toyota”, “audi”, “bmw”];  Đổi giá trị trong mảng: cars[0]=”merc”; // OK  Thêm hoặc bớt giá trị trong mảng: .shift( ) | unshift( ) | .pob( ) | .push( ) | .splice( ) | //OK  Sắp xếp vị trí giá trị trong mảng: .sort( ) | .reverse( ) | //OK  **Assigned when Declared:** Phải gán giá trị ngay khi khai báo  Nếu không const myArr; //myArr = undified;  ( với **var** thì gán sau khi khai báo thoải mái)  **Const Block Scope:** Const có phạm vi khối - const Arr ngoài **{ }** và trong **{ }** là khác nhau | | | | |
| **Date** | | **Ngày** | | |
| **Tạo một Date Object:**   * **new Date( ); - Thời gian hiện tại**   vd: const date = new Date(); // = Fri Sep 03 2021 21:41:23 GMT+0700 (Giờ Đông Dương)   * **new Date( year , month, day, hours, minutes, seconds, milliseconds)**   **\*\* Lưu ý: JavaScript đếm tháng từ 0 đến 11 // January = 0 ,…. , December = 11**  **Các năm 19xx có thể viết tắt bằng: xx**  **Các khai báo lớn hơn sẽ không gây lỗi nhưng sẽ được cộng dồn vào chỉ số sau.**  **Sử dụng 6, 4, 3 hoặc 2 số để khai báo Date():**  **6 số : new Date( year , month, day, hours, minutes, seconds )** vd: new Date(2021,8,3,21,50,30)  **4 số : new Date( year , month, day, hours)**  **3 số : new Date( year , month, day)**  **2 số : new Date( year , month)**   * **new Date( milliseconds) – Ngày có số mili giây cách mốc [ 0h ngày 01/01/1970 ]** * **new Date( date String ) -**   **Javascript Date Formats:**   * **ISO Date: “2015-03-25”** * **Short Date: “03/25/2015”** * **Long Date: “Mar 25 2015” hoặc “25 Mar 2015”** | | | | |
| **Math** | | **Toán học** | | |
| **Boolen** | | **True or False** | | |
| **Dùng hàm Boolean( ) để xác định biểu thức hoặc biến là true hoặc false**  **Tất cả mọi thứ không có giá trị đều là FALSE:**  **let x;**  **let x = 0;**  **let x = “”;**  **let x = null;**  **let x = false;**  **let x = 10/”abc”; 🡺 x = NaN;** | | | | |
| **Comparison** | | **Sự so sánh** | | |
| **Lưu ý khi so sánh 2 loại giá trị khác nhau:**   |  |  |  | | --- | --- | --- | | **Trường Hợp** | **Boolean** | **Giải thích** | | **2 < “12”** | **True** | **vì 2 == “2” mà “2” < “12” nên đúng** | | **2 > “”** | **True** | **vì “” chuyển sang Number là 0** | | **2 < “John”** | **False** | **“john” chuyển sang dạng Number sẽ là NaN**  **Bất cứ gì liên quan tới NaN đều là False** | | **2 > “John”** | **False** | | **2 == “john”** | **False** | | **“2” > “12”** | **True** | **vì “2” > “1”** | | | | | |
| **Loop FOR** | | **Vòng lặp for** | | |
| **Vòng lặp FOR …. IN - Lặp qua các KEY của một Object:**  **for( key in object){** vd:const person = { fname: “Binh” , lname: “Tran” , age:25}  **…. object[key]…..}** let text = “ ”;  **}** for ( let x in person){  text += person[x];  }  - Vòng lặp for…in lặp qua toàn bộ Properties của Object  - Mỗi lần lặp trả về một key  - key dùng để truy xuất giá trị của key đó  - Giá trị của key là object[key]  **Vòng lặp FOR …. OF - Lặp qua các VALUE của một Object có thể lặp lại:**  Cho phép lặp qua các cấu trúc dữ liệu có thể lặp lại như Array, String, Map, NodeList,…  **for( variable of iterable){ . variable:** có thể khai báo biến với const, let , var  // code block to be excecuted **. iterable:** Một Object có thuộc tính có thể lặp lại (iterable)  **}**  **FOR … OF trong Array: FOR … OF trong String**  const cars = [“bmw”, “Volvo”, “Mini”]; let language = “JavaScript”  let text = “ ”; let text=””;  for(let x of cars){ for( let x of language){  text += x + ‘ & ’; text += x +”-“;  } }  // text =” bmw & Volvo & Mini &” // text = “ J-a- v- a- S- c- r- I- p- t-“ | | | | |
| **Sets** | | **Một tập hợp các giá trị DUY NHẤT** | | |
| **Set is a collection of UNIQUE values** *Set là một tập hợp các giá trị DUY NHẤT*  **Each value can only occur once in a SET.** *Mỗi giá trị chỉ xuất hiện 1 lần trong Set*   * **Create a SET – Dùng new Set( ) :**   **- Thêm một mảng vào new Set( );**  const letters = **new Set**( [“a”, “b”, “c”, “d”] );  **- Khai báo một new Set( ) và dùng method .add( ) để thêm Unique Value vào set:**  const letters = new Set( );  letters.**add**( “a” ); // thêm trực tiếp giá trị vào  const b = “b”; letters.**add**( b ); // thêm biến vào   * **Set Property:** * **.size – Số lượng elements có trong 1 Set**   const letters = new Set( [ “a”, ”b”, ”c” ] );  letter.size; //=3   * **Set Methods:** * **.add( ) - Thêm value vào Set:**   const letters = new Set( [ “a”, ”b”, ”c” ] );  letters.add( “d” ); // letters.size = 4  letters.add( “a” ); // letters.size = 4; -**Khi thêm nhiều value giống nhau chỉ 1 cái được thêm**   * **.delete( ) – Xoá value khỏi set**   letters.delete(“a”); // letters.size = 3   * **.has( ) – Trả về True hoặc False nếu value có trong Set**   letters.has(“a”); // false   * **.forEach( ) – Gọi hàm cho mỗi giá trị trong Set**   let text= “ “;  letters.forEach( function(value){ return text += value;} ) // text=” b c d”   * **.values( ) – Trả về *một trình lặp lại* với tất các các giá trị trong Set ( Iterator )**   **- Set là 1 tập hợp không có thuộc tính có thể lặp lại, do đó nếu muốn lặp qua các giá trị trong Set ta có thể dùng forEach( ) hoặc dùng FOR … OF với Set.value( )**  const letters = new Set( [ “a”, ”b”, ”c” ] );  let text = “-“;  Không có .values() : for( let x of **letters** ){ text += value;}; // text = “-abc”; ( không lặp)  Có .values() : for( let x of **letters.value()** ){ text += value;}; // text = “-a-b-c”; | | | | |
| **Maps** | | **Bản đồ các cặp khoá và giá trị** | | |
| **A Map giữ các cặp KEY – VALUE, trong đó KEY có thể là bất cứ loại dữ liệu nào**  **A Map ghi nhớ thứ tự chèn ban đầu của các KEY**   * **Create a MAP – Dùng new Map( ) :**   **- Thêm một Array dạng Array trong Array vào trong Map:**  const fruits = **new Map( [** ["apples", 500], ["bananas", 300], ["oranges", 200] **] );**  **- Tạo một Map và sử dung Map.set( *key,value* ) :**  const fruits = new Map( );  fruits.**set**("apples", 500); //fruits = Map[ ["apples", 500] ];  fruits.**set**("bananas", 300); //fruits = Map[ ["apples", 500] , ["bananas", 300] ];  fruits.**set**("oranges", 200); //fruits = Map[ ["apples", 500], ["bananas", 300], ["oranges", 200]];   * **MAP Property :** * **.size** – **Trả về số lượng elements có trong map**   fruits.size; //=3   * **MAP Methods:** * **.set( ) – Thêm một cặp key-value vào map HOẶC thay đổi value cho một key**   const fruits = **new Map( [** ["apples", 500], ["bananas", 300], ["oranges", 200] **] );**  fruits.**set**(“bananas”,100); //fruits = Map**[** ["apples", 500], ["bananas", 100], ["oranges", 200] **]**   * **.get( ) – Lấy value của key trong Map**   fruits.**get**(“apples”); //= 500   * **.delete( ) – Xoá một element dựa vào key**   fruits.**delete**(“apples”); // fruits = Map[ ["bananas", 100], ["oranges", 200] ]   * **.has( ) – Trả về True / False nếu key có tồn tại trong Map**   fruits.**has**(“apples”); // False   * **.forEach( ) – Gọi một hàm cho mỗi cặp Key-Value trong Map**   let text = “” ;  fruits.**forEach**( function(value, key){  text += key + “=” + value;})   * **.entries( ) – Trả về *một trình lặp lại* với các cặp Key-Value trong Map**   let text = “” ;  **for**( const x **of** fruits.entries( ) ){ text += x}; // text = "bananas, 100”+"oranges, 200”  **for**( const x **of** fruits.values( ) ){ text += x}; // text = " 100”+" 200”  **for**( const x **of** fruits.keys( ) ){ text += x}; // text = " bananas”+" oranges” | | | | |
| **Typeof** | | **Loại dữ liệu** | | |
| **5 Data Type can contain values:**  **String | Number | Boolean | Object | Function**  **2 Data Type cannot contain value:**  **null | undefined**  **6 Type of Objects:**  **Object | Date | Array | String | Number | Boolean**   * **Typeof – Tìm Data Type của biến**   typeof "" // 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"  typeof function () {} // Returns "function"  typeof myCar // Returns "undefined" (nếu chưa gán giá trị cho myCar)  typeof **null**  // Returns "object"   * **Constructor Property -** **Tìm** **Hàm Tạo của biến**   "John".constructor // Returns function String() {[native code]}  (3.14).constructor // Returns function Number() {[native code]}  false.constructor // Returns function Boolean() {[native code]}  [1,2,3,4].constructor // Returns function Array() {[native code]}  {name:'John',age:34}.constructor // Returns function Object() {[native code]}  new Date().constructor // Returns function Date() {[native code]}  function () {}.constructor // Returns function Function(){[native code]}   * **null & undifined**   null == undifined // false  null === undifined // true | | | | |
| **Type Conversion** | | **Chuyển đổi dữ liệu** | | |
| **Có 6 loại chuyển đổi:**  **Strings 🡪 Numbers | Numbers 🡪 Strings | Dates 🡪 Numbers | Numbers 🡪 Date |**  **Booleans 🡪 Numbers | Number 🡪 Booleans**  **Có 2 cách để chuyển:**  **- Sử dụng Javascript function**  **- Javascript tự động chuyển đổi**   * **String 🡪 Numbers:** * **Global Method: Number( )** * **Number Methods: Number ( ) | parseInt( ) | parseFloat( )**   >> Number(“9 10”) 🡪 NaN -- Không được có khoảng trắng  >> parseInt(“9 10”) 🡪 9 -- Được có khoảng trắng và chọn số đầu tiên   * **The Unary + Operator: - Thêm toán tử + trước một biến String:**   >> let y = “5”; // typeof y : String  let x = + y; // typeof x : Number   * **Numbers 🡪 String:** * **Global Method: String( )**   >> String(123); //=”123”   * **Number Methods: .toString( ) | .toExponential( ) | .toFixed( ) | .toPrecison( )**   >> (123).toString( ); //=”123”   * **Dates 🡪 Numbers:** * **Global Method: Number( )** * **Date Method: getTime( )** * **Dates 🡪 String :** * **Global Method: String( )** * **Date Methods: .toString( ) |** các phương thức **.get…** * **Booleans 🡪 Numbers:** * **Global Method: Number( )**   >> Number(true) //=1  >> Number(false) //=0   * **Booleans 🡪 String:** * **Global Method: String( )**   >> String(true); //=”true”  **\*\*Automatic String Conversion:**  **Javascript tự động gọi hàm toString( ) khi bạn output một Object hay một biến.**  >> document.getElementById(“demo”).innerHTML = myVar;  // if myVar = { } // toString converts to “[object Object]”  // if myVar = [“a”,”r”,”r”] // toString converts to “a,r,r”  // if myVar = new Date() // toString converts to “Fri Jul 18 2014 09:08:55 GMT+0700 “ | | | | |
| **Bitwise** | | **Các phép toán Bitwise** | | |
|  | | | | |
| **RegExp** | | **Biểu thức chính quy** | | |
| **Cú pháp: / pattern /modifiers;**  **Các Methods sử dụng RegExp:**   * **.search( ) – Trả lại vị trí của RegExp**   let text = "Visit W3Schools";  let n = text.**search**(/w3schools/i); // n = 6   * **.replace( ) – Thay thế vị trí của RegExp**   let text = "Visit W3Schools";  let result = text.**replace**(/w3schools/i , “Microsoft” ); // result = “Visit Microsoft”  **Các Method của RegExp Object:**   * **.test( ) – Trả về kết quả True / False từ việc tìm kiếm RegExp**   /e/.**test**("The best things in life are free!"); //=true;   * **.exec( ) – Trả về các RegExp tìm thấy đưới dạng một Object, nếu không thấy trả về null**   const obj = /e/.**exec**("The best things in life are free!");  obj[0] = e  obj.index = 2  obj.input = "The best things in life are free!"); | | | | |
| **Errors** | | **Ném và bắt lỗi** | | |
| **try – Cho phép kiểm tra một khối code để tìm lỗi**  **catch – Cho phép xử lý lỗi**  **throw – Cho phép tạo lỗi do người dùng định nghĩa**  **finally – Cho phép thực thi mã, sau khi try và catch bất chấp kết quả**   * **try and catch Error:**   **Cú pháp: try {**  *block of code to try*  **}**  **catch {**  *block of code to handle errors* // câu lệnh được thực thi khi có lỗi xảy ra trong **try**  **}**  **Example:**  <p id=”demo”> </p>  <script>  **try** {  addlert( “Hello World! “ ); “Lỗi sai cú pháp **alert**  } Lúc này JavaScript **throw** ra một Error Object  **catch**(err){ Và ta dùng **catch** để bắt lấy nó”  document.getElementById(“demo”).innerHTML = err.message; //err.message = addlert is not defined  }  </script>   * **throw Error**   **-** Khi một lỗi xảy ra, JavaScript thường sẽ dừng lại và phát ra một ERROR Message  Quá trình này gọi là throw Error  - Quá trình này tạo ra một Error Object có 2 thuộc tính là **name** và **message**   * **throw Statement**   Cho phép bạn tạo một custom Error , có thể throw a String, a Number, a Boolean, a Object.  vd: **throw** “Số quá lớn”; // throw a text  **throw** 500; // throw a Number  **Các giá trị throw sẽ thay thế cho Erros Object và dùng catch để bắt nó;**  vd:  const x = 6;  **try**{  If( x == 6 ) **throw** “chinh xac”; if ( x < 6) **throw** “nho hon”; if ( x > 6) **throw** “lon hon”;  }  **catch**(err){  document.getElementById(“demo”).innerHTML = err ;  }   * **HTML Validation**   Các trình duyệt ngày nay thường sử dụng kết hợp JavaScript và Xác thực HTML, sử dụng các qui tắc xác thực trong thuộc tính HTML:  vd : <input id="demo" type="number" min="5" max="10" step="1">   * **The finally Statement - Một statement luôn được thực thí không liên qua try-catch**   vd: **try** { }  **catch** { }  **finally** {  alert(“ qua trình hoàn tất” );  }   * **The Error Object** * **Error name:**   **catch(err){ document.getElementById(“demo”).innerHTML = err.name ;}**   **- EvalError – Đã xảy ra lỗi trong hàm Eval( )** // New JavaSript sử dụng SyntaxError thay thế  **- RangeError – Một Number nằm ngoài phạm vi** vd: toPrecision(500) mà tối đa là 100  **- ReferenceError – Một tài liệu tham khảo không hợp lệ** vd: sử dụng biến chưa khai báo  **- SyntaxError – Sai cú pháp** vd: sai chính tả, thiếu dấu  **- TypeError – Lỗi Type** vd: (1).toUpperCase  **- URIError – Lỗi trong việc mã hoá URI trong encodeURI( )** vd: decodeURI(“%%%”)   * **Error message:** catch(err){ document.getElementById(“demo”).innerHTML = **err.message** ;}   **Trả về một thông báo lỗi dạng String**   |  |  |  | | --- | --- | --- | | **Example** | **err.name** | **err.message** | | **var num = 1;**  **try{ num.toPrecision(500); }** | **RangeError** | toPrecision() argument must be between 1 and 100 | | **let x = 5;**  **try{ x = y + 1;}** | **ReferenceError** | y is not defined | | **try{ eval( “alert( ‘Hello ) ” ) ; }** | **SyntaxError** | Invalid or unexpected token | | **let num = 1;**  **try { num.toUpperCase(); }** | **TypeError** | num.toUpperCase is not a function | | **try{ decodeURI( “ %%% “ ); }** | **URIError** | URI malformed // dị dạng |   **.** | | | | |
| **Scope** | | **Phạm vi** | | |
| **Javascript have 3 types of Scope:**   * **Function Scope - Phạm vi hàm** * **Block Scope – Phạm vi khối** * **Global Scope – Pham vị toàn bộ**   **- Biến khai báo với let và const có Block Scope :**  Biến khai báo bên trong { } không thể truy xuất bên ngoài khối  // x can NOT be used here // x can NOT be used here  **{** **{**  let x =2 ; const x = 2;  **}** **}**  // x can NOT be used here // x can NOT be used here  **- Biến khai báo với var không có Block Scope:**  Biến khai báo bên trong { } có thể truy xuất bên ngoài khối   * **Function Scope:**   Các biến khai báo bên TRONG hàm không thể truy cập được từ bên ngoài hàm  Các biến khai báo với var, let, const tương tự nhau khi khai báo trong hàm và đều có Function Scope.  - Khi gọi một hàm luôn có một phạm vi mới được tạo  - Các hàm có thể truy cập các biến được khai báo bên ngàoi hàm đó và bên trong phạm vi của nó.   * **Global Variable:**   Các biến khai báo bên NGOÀI hàm trở thành Global và có thuộc tính Global Scope.   * **Global Scope:**   Các biến có tính Global Scope có thể truy xuất tại bất cứ đâu trong JavaScript   * **Automatically Global: ( không hiệu lực trong Strick Mode)**   Khi bạn gán một giá trị cho một biến **CHƯA ĐƯỢC KHAI BÁO**, nó sẽ tự động trở thành biến **Global**  myFunction(); //gọi hàm kích hoạt biến  // code here can use carName  function myFunction( ){  carName = “volvo”;  } | | | | |
| **Hoisting** | | **Tự động đưa khai báo lên trên** | | |
| **< Hoisting không sử dụng trong chế độ STRICK MODE – “use strick” >**   * **Declarations are Hoisted – Khai báo được Hoisted**   x = 5; // Assign 5 to x  elem = document.getElementById("demo"); // Find an element  elem.innerHTML = x; // Display x in the element  var x; // Declare x  **\*\* Các biến được khai báo với let và const cũng được hoisted lên top nhưng KHÔNG được khởi tạo\*\***  carName = “volvo”; 🡪 err.message = “Cannot access 'carName' before initialization”  let carName;  carName = “volvo”;  const carName;  **Nhắc lại qui tắc let và const:** **let** – phải khai báo trước rồi có thể gán sau  **const** – phải khai báo + gán trước, ( nếu không gán nó là undifined)   * **Initialization are NOT Hoisted – Khởi tạo không được Hoisted (//** khởi tạo = khai báo + gán**)**   var x = 5; // Initialize x  elem = document.getElementById("demo"); // Find an element  elem.innerHTML = x + " " + y; // Display x and y 🡪 Kết quả : x undifed  var y = 7; // Initialize y  **( chỉ có var y được hoisted còn y = 7 không được hoisted)** | | | | |
| **STRICK MODE** | | **Chế độ chính xác** | | |
| **Thêm “use strict” vào đầu script hoặc function để kích hoạt**  <script>  x = 3.14; // không gây lỗi  myFunction();  function myFunction() {  “use strict”  y = 3.14; // gây lỗi vì chưa khai báo biến x  }  </script> | | | | |
| **this** | | **Từ khoá this** | | |
| **this đại diện cho Object mà nó thuộc về:**   * **Đứng một mình, this đại diện cho Global Object**   let x = **this**; // this is [ object Window]   * **Trong Function, this đại diện cho Global Object**   function myFunction(){  return **this**; // this is [object Window]  }   * **Trong Function, ở chế độ STRICK MODE, this là undefined**   **“use strick”**  function myFunction(){  return **this**; // this is undefined  }   * **Trong Methods, this đại diện cho Objects sở hữu method đó**   const **obj** = {  …... ,  fullName : function( ) { return **this**.firstName + “ “ + **this**.lastName; } // **this** is **obj**  }   * **Trong Event, this đại diện cho element được nhận event đó**   **<button** onclick=” **this**.style.display=’none’ ” > Click **</button>** // **this** is **<button></button>**   * **Các Methods như call( ) và apply( ) có thể qui this cho bất cứ object nào.**   \*\* call( ) và apply( ) là những Methods được định nghĩa trước  Dùng để sử dụng gọi một Object Method với một Object khác như là đối số  const **person1** = {  fullName: function() { return **this**.firstName + " " + **this**.lastName; }  }  const **person2** = {  firstName:"John",  lastName: "Doe",  }  person1.fullName.**call(**person2); // Will return "John Doe" -- ở đây **this** is **person2** | | | | |
| **Arrow Function** | | **=> Function** | | |
| **Arrow Function dùng để viết ngắn Syntax cho Function:**  **Before: hello = function() { After: hello = ( ) => {**  **return "Hello World!"; return "Hello World!";**  **} }**  **Nếu Function chỉ có 1 câu lệnh và Return value - Ta có thể bỏ { } và return:**  **hello = ( ) => “Hello World!”**  **Nếu có tham số :**  **hello = ( *parameter* ) => “Hello World!” + *parameter*;**  **Hoặc Bỏ luôn ( ):**  **hello = *parameter* => “Hello World!” + *parameter*;**  **this trong Arrow Function:**  **- Ở regular Function this đại diện cho Object gọi hàm Function đó, có thể là window, document, button,….**  **hello = function() {**  **document.getElementById("demo").innerHTML += this;**  **}**  // The window object calls the function:  **window.addEventListener("load", hello); // tại đây this là Window Object**  // A button object calls the function:  **document.getElementById("btn").addEventListener("click", hello); // tại đây this là Button Object**  **- Trong Arrow Function, this LUÔN LUÔN đại điện cho Object đã định nghĩa Arrow Function.**  **hello () => {**  **document.getElementById("demo").innerHTML += this;**  **}**  // The window object calls the function:  **window.addEventListener("load", hello); // tại đây this là Window Object**  // A button object calls the function:  **document.getElementById("btn").addEventListener("click", hello); // tại đây this là Window Object** | | | | |
| **Classes** | | **Các lớp JavaScript** | | |
| **JavaScript Classes là các MẪU cho Javascript Objects;**  **Sử dụng từ khoá class để tạo một class**  **Luôn luôn thêm phương thức tên constructor( ) chứa các Props**  **Syntax:**  **class className {**  **constructor(){……..}**  **method\_1(){ ……..}**  **method\_2(){………}**  **……………..**  **}**  **>> Class không phải là Object nó mà Template (mẫu) của Object <<**  **>> Có thể sử dụng Class để tạo Object <<**  **So sánh Javascript Class với Object Constuctor**   |  |  | | --- | --- | | **class – Objects Template** | **Object constructor** | | class Car {  constructor(name, year) {  this.name = name;  this.year = year;  }  age() {  let date = new Date();  return date.getFullYear() - this.year; }  }  let myCar = new Car(“Ford”,2014)  **🡪 car.age(); = 7** | function Car(name,year){  this.name = name;  this.year = year  this.age = function(){  let date = new Date();  return date.getFullYear() - this.year;  }  }  let myCar = new Car(“Ford”,2014)  **🡪 car.age(); = 7** | | | | | |
| **JSON** | | **JavaScript Object Notation** | | |
| **- JSON - JavaScript Object Notation - Ký hiệu đối tượng trong JavaScript**  **- Là một định dạng chuyển đổi dữ liệu**  **- JSON là một ngôn ngữ độc lập**  **- JSON là “tự mô tả” và dễ để hiểu**  // Kiểu JSON là một đoạn text có thể dùng để đọc và tạo dữ liệu bằng bất cứ ngôn ngữ lập trình nào  **JSON qui tắc cú pháp:**   * **Data là các cặp name – value; name nằm trong dấu ngoặc kép “” - “name”** * **JSON có thể thể hiện được dữ liệu dạng : Number, String, Array, Object, Boolean, Null** * **Data được tách ra bằng dấu phẩy ,** * **Object nằm trong { }** * **Array nằm trong [ ]**   vd**: JSON ARRAY: “employees” : [**  **{ “firstName” : “John” , “lastName” : “Doe” } ,**  **{ “firstName” : “Anna” , “lastName” : “Smith” } ,**  **{ “firstName” : “Peter” , “lastName” : “Jones” } ,**  **]**  **Chuyển đổi JSON TEXT thành JavaScript Object :**  - Thông thường JSON được sử dụng để đọc dữ liệu từ Web Server và hiển thị dữ liệu ở Web Page;  - Đầu tiên , tạo một JavaSript String chứa cú pháp JSON:  *Sử dụng Back-tics:* **let text = `{** **“employees” : [**  **{ “firstName” : “John” , “lastName” : “Doe” } ,**  **{ “firstName” : “Anna” , “lastName” : “Smith” } ,**  **{ “firstName” : “Peter” , “lastName” : “Jones” } ,**  **] }` ;**  **- Sau đó dùng hàm built-in của JavaScript JSON.parse( ) để chuyển JSON String thành Object:**  **const obj = JSON.parse( text );**  **// Lúc này trong JavaScrip đã có một Object obj:**  **obj = { employees : [ { firstName : “John” , lastName : “Doe” } ,**  **{ firstName : “Anna” , lastName : “Smith” } ,**  **{ firstName : “Peter” , lastName : “Jones” }**  **] };**  obj.employees[0].firstName; = “John”;  - **JSON.stringtify : --> Chuyển từ JavaScript sang JSON** | | | | |
| **STYLE GUILDE** | | **Qui tắc kiểu dáng trong JavaScript** | | |
| * **Variable Names: bắt đầu với một chữ viết thường và các chữ sau viết HOA chữ cái đầu:** * **Spaces Around Operators : luôn có khoảng trắng xung quanh các toán tử** * **Code Indentation : Sử dụng 2 space để thụt dòng code** * **Statement Rules: Các qui tắc câu lệnh:**   **Đối với câu lệnh đơn giản :**   * **Luôn kết thúc câu lệnh với dấu chấm phẩy ;** vd: const car = [“Volvo”,Saab”];   **Đối với câu lệnh phức tạp :**   * **Đặt dấu mở ngoặc { và [ ở cuối dòng dầu tiên** * **Sử dụng một khoảng trắng trước dấu mở ngoặc** * **Đặt dấu đóng ngoặc } và ] ở một dòng mới và không có khoảng trắng trước nó** * **Không kết thúc một complex stament – câu lệnh phức tạp với dấu chấm phẩy**   **vd:** function sum(a,b) {  return (a + b) ;  }   * **Line Length < 80 – Chiều dài mỗi dòng không quá 80 ký tự** * **Loading JavaScript in HTML – The TYPE attribute is not necessary**   <script src=”myscript.js” ></script> | | | | |
| **CALLBACK FUNCTION** | | | **HÀM CALLBACK** | |
| **- Callback là hàm được truyền qua đối số cho một hàm khác và được gọi lại trong hàm đó**  **vd:**  **function myFunction1(***parameter1***) {**  //execute with parameter1  **}**  **function myFunction2(myCallback) {**  **myCallback(** *parameter2* **);**  **}**  **myFunction2(myFunction1);**  //truyền hàm **myFunction1** như là một đối số vào **myFunction2**;  lúc này hàm **myCallback** có giá trị là **myFunction1**  \*\* ***Lưu ý: khi truyền một hàm như là một đối số ta không sử dụng toán tử ( );*** | | | | |
|  | | | | |
| **Promise Object** |  | | | |
| **- Promise dùng để giải quyết vấn đề callback hell ( các hàm phụ thuộc vào hàm khác)**  **- “Producing code” - Là mã cần Thời Gian để thực hiện**  **- “Consuming code”- Là code cần chờ đợi Kết Quả**  **- “Promise” là một Object kết nối “Producing code” và “Consuming code”.**  **let *myPromise* = new Promise( function(*myResolve*, *myReject*) {**  ***myResolve*( value );** *// when successfull*  ***myReject*( error );** *// when error*  **} );**  **myPromise.then(**  **function( value ) { …***code if successful…* **} ,**  **function( error ) { …***code if some error…* **}**  **) ;**  **\*\* Promise Property:**  **- Object Promise có 2 thuộc tính là .state** ( trạng thái ) **và .result** (kết quả) để theo dõi   |  |  |  | | --- | --- | --- | | **myPromise.state** | **myPromise.result** | **Mô tả** | | **“pending”** | ***undifined*** | Khi Promise đang chạy | | **“fulfilled”** | ***a result value*** | Khi Promise đang chạy thành công | | **“rejected”** | ***an error object*** | Khi Promise không chạy được |   **\*\* Promise Method:**  **- .then( function(value, error) { } )** là một hàm nhận 2 đối số là **value** và **error** để truyền vào các hàm callback để xử lý.  vd: myPromise.then( function(value) {  // code with result- value //  } );  - **.catch(function(error) { })** là một hàm nhận **error**  - .**finally( function() { } )** là một hàm luôn chạy kể cả khi có error xảy ra  **>> các phương thức của Promise mặc định:**  **let myPromise = Promise.resolve( result** ); // tạo một Promise luôn chạy thành công  **let myPromise = Promise.reject( error );** // tạo một Promise luôn chạy thất bại  **let myPromise = Promise.all( [promise1 , promise2 ] );**  // tạo một Promise chờ 2 Promise con chạy song song nhau  **\*\*\* Tính chất CHAIN của promise: // tính chất làm việc asyn đồng bộ**  **vd: myPromise**  **.then( function(value) {** ….. return value **} )**  **.then( function(value) {** ….. return value **} )** // những thằng then sau sẽ đợi value của  **.then( function(value) {** ….. return value **} )** thằng then trước  **.then( function(value) {** ….. return value **} );**  **.catch(function(error) { })**  // nhận error của myPromise  **🡺 Nếu trong then return là một Promise thì những thằng then sau sẽ là là then của Promise đó**  **myPromise**  **.then( function(value) {** ….. return **new Promise** **} )**  **.then( function(value) {** ….. return value **} )** // những thằng then sau sẽ đợi value của  **.then( function(value) {** ….. return value **} )** thằng then trước  **.then( function(value) {** ….. return value **} );** | | | | |
| **Async / Await** | **Không đồng bộ và Chờ** | | | |
| **Async :** Khiến một function trở thành PROMISE  **Await :** Khiến một function chờ một PROMISE ;  **- Từ khoá await chỉ sử dụng đươc trong hàm có sử dụng từ khoá async** | | | | |
| **API** | **Application programing interface** | | | |
| **API - Giao diện lập trình ứng dụng:**  **- Là một cổng giao tiếp giữa font-end và back-end , hoặc giữa các ứng dụng với nhau**  **- Backend --> API --> Fetch --> JSON/XML … --> Jason.parse( ) để nhận Javascript Type**  **vd: placeholder** | | | | |
| **fetch** | **Gọi Api lấy dữ liệu** | | | |
| **- fetch(‘url’) --> sẽ trả về một *Promise* là stream chứa dữ liệu**  **vd: fetch( ‘** [**https://jsonplaceholder.typicode.com/posts**](https://jsonplaceholder.typicode.com/posts) **’ ) //**  **.then( function(*response*) {**  **return *response*.json() ; //** Phương thức .json() trả về một *Promise* là **JSON.parse**  **}**  **.then( function( *posts* ) {**  **console.log(*posts*)** // lúc này post nhận được có dạng Javascript Type  **}**  **fetch sử dụng 2 tham số : fetch( ‘url’ , option )**  **Trong đó option là một Object chứa các cách thức hoạt động của fetch:**  **option = {**  **method : ‘POST’ , ( ‘GET’ | ‘PUT’ | ‘PATCH’ | ‘DETELE’)**  **header: {‘Content-type’ : ‘application/json’ } , ( x-www-form-urlencoded )**  **body : JSON.stringify(*data*)**  **}**  **- Với method: GET , fetch( ‘url’, option) trả về một stream chứa toàn bộ nội dung của API**  **- Với method: POST , fetch( ‘url’, option) trả về một stream chứa nội dung của API được tạo mới**  **- Với method: PUT , fetch( ‘url’, option) trả về một stream chứa toàn bộ nội dung của API được tạo sửa**  **( gồm các key-value mới cùng với id)**  **- Với method: PATCH , fetch( ‘url’, option) trả về một stream chứa toàn bộ nội dung của API được sửa**  **( gồm các key-value cũ và mới cùng với id)**  **- Với method: DELETE , fetch( ‘url’, option) trả về một stream rỗng** | | | | |
| **JSON server** | **Tạo một api giả** | | | |
| **- JSON server: API server ( fake API) hay còn gọi là Mock API - dùng làm giả cách thức làm việc với API bằng các dữ liệu tự tạo trên localhost**  **1. Tạo dữ liệu bằng 1 file name\_file.json**  **vd: db.json**  **Lưu các dữ liệu cần dưới dạng json:**  **{**  **“myFakeAPI” : [{**  **“id” : 1, // bắt buộc có id để đánh dấu vị trí các object**  **“ hoten “ : “baobinh” ,**  **“ tuoi” : 18 ,**  **“mssv” : 1203045**  **}]**  **}**  **2. mở mục {}package.json:**  **dưới vị trí: “scripts”**  **thêm : “start”: “json-server --watch db.json”,**  **3. Trong terminal gõ lệnh: npm *start***  **ta sẽ nhận được một API với url sau: http://localhost:3000/ myFakeAPI**  **có chứa nội dung json trong db.json** | | | | |
| **Postman** | **Làm việc với REST API** | | | |
| **- CRUD: các hành động thao tác với dữ liệu**  **- Create : Tạo mới 🡺 POST**  **- Read: Lấy dữ liệu 🡺 GET**  **- Update: Chỉnh sửa 🡺 PUT / PATCH**  **- Delete: Xoá 🡺 DELETE**  **- Postman : Phần mềm dùng để thực hiện các tác vụ với API giả**  **Hướng dẫn sử dụng Postman:**   * **Get: http://localhost:3000/ myFakeAPI**   - Chọn phương thức Get , dán url FAKE API từ nodeJd Json Server  vd: GET **http://localhost:3000/ myFakeAPI**   * **Post: http://localhost:3000/ myFakeAPI**   Vào mục Body , chọn x-www-form-urlencoded  Thêm các mục mới vào Key - Value  Rồi ấn SEND  Sẽ tạo được một object mới nằm trong mảng dữ liệu với id tăng lên   * **Put / Patch: http://localhost:3000/ myFakeAPI / id**   Thêm id của phần muốn sửa vào cuối URL để sửa  Với PUT:  Phải gửi một bản hoàn chỉnh gồm tất cả nội dung của id đó, bao gồm các key-value không sửa  Nếu không nó sẽ bị xoá  Với PATCH:  Chỉ cần gửi các cặp Key - Value muốn sửa, những key-value khác sẽ vẫn giữ nguyên   * **Delete: http://localhost:3000/ myFakeAPI / id**   Thêm id của phần muốn xoá vào cuối URL để xoá | | | | |
| **Rest Parameters** | **Toán tử rest** | | | |
| **>> Dùng toán tử … với biến ta gọi đó là REST <<**  **function logger( … params) { //Không giới hạn số lượng parameter truyền vào**  **console.log(params);**  **}**  **console.log( logger(1,2,3,4,5,6) ); // Trả về nhiều giá trị ở dạng mảng [1,2,3,4,5,6]** | | | | |
| **Destructuring** | **Sự phá huỷ** | | | |
| **- Destructuring với Mảng**  **var myArray = [ ‘Javascript’, ‘PHP’, ‘Ruby’ ];**  **Đặt các biến muốn lấy vào vị trí biến cần lấy**  **var [ a, b, c] = myArray;** // lúc này a = ‘Javascript’ ; b = ‘PHP’, c = ‘Ruby’  **var [ a, , ] = myArray;** // lúc này a = ‘Javascript’ ;  Toán tử **rest … phần còn lại**  **var [ a, …d] = myArray;** // lúc này a = ‘Javascript’ ; d = [‘PHP’, ‘Ruby’ ];  **- Destructuring với Object:**  **const course = {**  **name: ‘Javascript’,**  **price: 1000,**  **decription: ‘Khoá học’**  **}**  Phải đặt đúng các key vào vị trí cần lấy:  **var {name, price, decription } = course;** // lúc này name= ‘Javascript’, price=1000, decription= ‘Khoá học’  Có thể đặt tên biến bằng các thêm dấu hai chấm : sau các key :  **var {name : ten, price: gia, decription: mieuta } = course;**  // lúc này ten= ‘Javascript’, gia=1000, mieuta= ‘Khoá học’  Toán tử **rest … phần còn lại**  **var {name, … rest } = course;** // lúc này name= ‘Javascript’, rest = { price: 1000, decription: ‘Khoá học’ }  **Với Object nằm trong Object:**  **const course = {**  **name: ‘Javascript’,**  **children: {**  **name: ‘ReactJs’**  **}**  **}**  **var {name: ten , children: {name: ten2} = course;**  // lúc này ten= ‘Javascript’, ten2 = ‘ReactJs’  **Với giá trị Undefined:**  **const course = {**  **name: ‘Javascript’,**  **price: 1000,**  **}**  **var {name, price, decription } = course;**  // lúc này decription = undefined; ta có thể gán giá trị vào cho nó  **var {name, price, decription = ‘default value’} = course;**  // lúc này decription = ‘default value’, và sẽ thay đổi nếu trong Object có gán giá trị cho decription | | | | |
| **Spread** | **Lan tràn** | | | |
| **Toán tử Spread . . . với Array:**  **var array1 = [ ‘a’ , ‘b’, ‘c’]**  **var array2 = [ ‘d’, ‘e’ ]**  **var array3 = [ …array2, …array1] ; // array3 = [ ‘d’, ‘e’, ‘a’ , ‘b’, ‘c’ ]**  **Toán tử Spread . . . với Object:**  **var object1 = {**  **name: ‘Javascript’,**  **price: 1000,**  **}**  **var object2 = {**  **decription: ‘Khoá học’**  **}**  **var object3 = { …object1 , …object2 }**  **>> Khi đặt toán tử Spread … trước Array hay Object sẽ bỏ đi dấu ngoặc { } và [ ] <<** | | | | |
| **Tagged template literals** | **Ký tự mẫu được gắn thẻ** | | | |
| **function highlight ( …rest) {**  **}**  **var brand = ‘F8’;**  **var course = ‘Javascript’;**  **highlight `Học lập trình ${ course } tại ${ brand } !` ;**  **🡺 Các tham số trong dấu back-tic sẽ truyền vào hàm highlight là một mảng như sau:**  **rest = [ [ “Học lập trình” , “ tại” , “!” ] , “Javascript” , “F8” ]**  **- Giá trị đầu tiên là một mảng các string không phải các giá trị nội suy ${}**  **- Các giá trị tiếp theo là các giá trị nội suy ${}**   function highlight([*first*,...*string*],...*value*){          console.log(first); // ‘đây là bài test          console.log(string); // [ “và”, “”]          console.log(*value*); // [“F8” , “Javascript”]          return *value*.reduce( function(*acc*,*curr*) {              return […acc,`<span>${*curr*}</span>`,string.shift()])          }, [first]).join(“”);      }      var brand = 'F8';      var course = 'Javascript';      let html = highlight`đây là bài test ${brand} và ${course}`;  // htmt = “đây là bài test <span> F8 </span> và <span> Javascript </span>” | | | | |
| **Modules** | **Import và Export** | | | |
| **Để nạp dữ liệu từ một file .js khác vào file js hiện tại**  **tại hmtl liên kết file js hiện tại: thêm type= “module” vào thẻ <script>**  **cú pháp: import *(cái gì đó)* form ‘./namefile.js’;**  **vd: import *name* form ‘./namefile.js’; // import default**  **import {…rest }form ‘./namefile.js’; // import distructuring ,**  **import \* as *name* form ‘./namefile.js’; // import *name* là một object chứa tất cả các export**    **Để xuất dữ liệu từ một file js:**  **cú pháp : export default *(cái gì đó)* ; // chỉ được 1 cái có default**  **export *(cái gì đó)* ;**  **export *(cái gì đó)* ;**  **hoặc có thể export từ một thằng js khác:**  **export { default } form ‘./namefile.js’**  **nó tương tự với: import *name* form ‘./namefile.js’ // nhận default name**  **export default name // xuất default** | | | | |
| **Optional chaining (?.)** |  | | | |
| **Vd: để kiểm tra trong một Object có key không:**  **const obj = {**  **key1 : {**  **key2 : {**  **key3: value**  **}**  **}**  **}**  **bình thường để kiểm tra ta dùng toán tử && như sau :**  **if ( obj.key1 && obj.key1.key2 && obj.key1.key2.key3) { }**  **Để gọn hơn ta dùng ?. : if (obj?.key1?.key2?.key3) { }**  **Có thể dùng với Array để kiểm tra index: if( array?.[index] ) { }**  **Có thể dùng với function để kiểm tra có tồn tại hay không: if( myFunction?.() )** | | | | |
| **DOM .bind()** | **Tạo viết tắt** | | | |
| **const $ = document.querySelector.bind(document) ;**  **const $$ = document.querySelectorAll.bind(document) ;**  **Lúc này $() sẽ tương tự document.querySelector()**  **và $$() sẽ tương tự document.querySelectorAll()** | | | | |
| **DOM .parentElement** | **Lấy ra thẻ cha của thẻ cần tìm** | | | |
| **Polyfill** | **Viết các tính năng hỗ trợ trình duyệt cũ** | | | |
|  | | | | |
|  | | | |
| **IIFE** | **Immediately Invoked Function Expression** | | | |
| **- Một biểu thức gọi ra một hàm và thực hiện hàm ngay lập tức**  **vd:**  **( function() {**  **console.log(‘do it’);**  **} )( )**  **--> khác với cách cũ là khai báo hàm xong mới gọi hàm đó như sau:**  **funtion myFunction() { console.log(‘do it’);**  **myFunction();**  **- IIFE là một “private” , Những gì khai báo bên trong ( ) không thể truy cập được từ bên ngoài.**  **--> Dùng làm các thư viện nhúng mà không làm ảnh hưởng đến code của người khác** | | | | |
| **Closure** |  | | | |
| **Khái niệm:**  **Closure : là một hàm có thể ghi nhớ nơi nó được tạo ra và truy cập được biến ở bên ngoài phạm vi của nó** | | | | |
| **Value Type** | **Tham trị** | | | |
| **Value Type : Tham trị - Primitive data types**  **Gồm các kiểu dữ liệu cơ bản:**  **-String**  **- Number**  **- Boolean**  **- undifined**  **- null**  **- BigInt**  **- Symbol**  **-- Với Tham trị, khi khai báo một biến ta sẽ tạo ra một ô nhớ cho nó và gán giá trị trực tiếp cho nó.**  **let a = 1;** 🡺 Tạo ra biến a, cấp một ô nhớ, lưu giá trị 1 vào ô nhớ đó;  **let b = a; 🡺** Tạo ra biến b, cấp 1 ô nhớ và gán giá trị 1 cho b;  **a = 2 ;** 🡺Gán giá trị 2 cho a;  **console.log(b); 🡺 1;** | | | | |
| **Reference Type** | **Tham chiếu** | | | |
| **Reference Type:**  **- Object**  **- Array**  **- Function**  **-- Với tham chiếu, khi khai báo một biến ta sẽ tạo ra một ô nhớ cho nó và gán địa chỉ của ô nhớ chứa nội dung được gán.**  **let a = { myObject1}**  **let b = a;**  **-- Lúc này cả a và b đều được gán địa chỉ ô nhớ chỉ tới ô nhớ có chứa giá trị { myObject }**  **Tổng kết:**  **Tham trị: gán trực tiếp giá trị cho biến**  **Tham chiếu: gán địa chỉ của ô nhớ chứa giá trị cho biến** | | | | |
| **.bind( )** |  | | | |
| **- Trả về một hàm mới và nhận các đối số như hàm ban đầu** | | | | |
| **.call( )** |  | | | |
| **- Là một phương thức trong prototype của Function constructor, phương thức này được dùng để gọi hàm và cũng có thể bind this cho hàm**  Bình thường khi ta dùng toán tử **( )** để gọi một hàm: **myFunction( );**  Lúc này Javascrip sẽ tự động sử dụng phương thức call( ) để gọi hàm: **myFunction.call( );**  **- Phương thức call( ) sẽ gọi hàm và bind( this );** | | | | |
|  |  | | | |
|  |  | | | |
|  |  | | | |