**Báo cáo JS**

1. JavaScript

- Là ngôn ngữ lập trình phổ biến nhất trên thế giới

- có thể thay đổi nd thẻ HTML, các giá trị thuộc tính, thay đổi style CSS, ẩn hiển thị các phần tử html

**2. Thẻ script**

- Mã đc chèn trong thẻ <script></script>

- function: là 1 khối mã, thực thi khi đc gọi

* đặt trong <head>
* or cuối <body> để tăng tốc độ hiển thị
* or external file đuôi .js, đặt tên file vào attribute src của <script>

**3.** **Khả năng hiển thị**

- Viết vào phần tử html: innerHTML

- Ghi vào đầu ra html: document.write() : used for testing

Sử dụng sau khi trang html đc tải thì sẽ xóa hết html

- Viết vào hộp cảnh báo: window.alert()

- Ghi vào bảng điều khiển của browser: console.log()

Window.print(): in nd cử sổ hiện tại

**4. Câu lệnh**

- Gồm: giá trị, toán tử, biểu thức, từ khóa, comment

**5. Khai báo biến** (k đc gán giá trị thì là undefined)

Bắt đầu: a-z, \_, $

- var: biến toàn cục, có thể khai báo lại

- let: k đc khai báo lại, có phạm vi khối

- const: hằng số, k thể thay đổi, có phạm vi khối

**Có 8 kiểu dl**:

Number, Bigint, String, Boolean

Null: trống or k tồn tại

Undefined: có giá trị duy nhất k đc xđ

Object, symbol: kiểu phức tạp

* Kiểm tra kiểu dl: typeof

**6. Một số hàm đc xây dững sẵn trong JS**

Alert, Console

Confirm: hộp thoại lựa chọn confirm, trả về true false

Prompt: hộp thoại nhập giá trị

SetTimeout: đoạn code chạy sau 1 khoảng tg

SetInrerval: đoạn code chạy sau 1 khoảng tg và lặp lại

**7. Toán tử**

\*\* lũy thừa

Prefix (tiền tố): ++a

Postfix (hậu tố): a++

?? Nullish: a ?? b return a nếu a kp null/underfined, nếu k, return b

== ss giá trị

=== ss giá trị, kiểu dl

* 6 giá trị khi convert sang boolean trả về false: 0, false, ‘’, underfined, NaN, null

**8. Function**

- hàm trong dòng mã chính: function <tên hàm>(biến){code}

- hàm trog ngữ cảnh của 1 biểu thức: let <tên hàm> = function(biến){code};

- arrow function: let <tên hàm> = (biến, …) => {return}

**9. Data types**

* ***Nguyên thủy***

- string, number, bigint, Boolean, symbol, null

***1. Number***: 2 loại

+ Các số phẩy động, lưu trữ dạng 64bit IEEE-754

+ Số BigInt: biểu thị số nguyên có độ dài tùy ý

NaN: số k hợp lệ

isNaN: check số có phải NaN k

***2. String***

- chuỗi đc mã hóa bằng UTF-16

+ *Tìm kiếm*:

- indexOf (chuỗi, vị trí bắt đầu): tìm kiếm từ 1 vị trí, trả về vị trí, trả về -1 nếu k thấy j

để trả về dạng Boolean: thêm ~ vào trước (đc tìm thấy)

- includes (chuỗi, vị trí bắt đầu): trả về boolean

- startWidth(), startEnd()

+ *Lấy chuỗi con*:

- slice (start, end), substring (start, end), subsrt (start, length)

Slice(0): copy

+ Chuyển chuỗi sang Array: split(‘điểm chung’)

* ***Object:*** lưu trữ nhiều giá trị dưới dạng thuộc tính

**10. Objects**

- Kiểu phức tạp, lưu trữ nhiều kiểu dl khác nhau

- Tạo Object: 2 syntaxes

* Obj constructor:

let <tên Obj> = new Object ();

* Obj leteral

let <tên Obj> = {

key: value,

};

Thêm: <tên Obj>.key=value;

Xóa: delete <tên Obj>.key;

Vs key nhiều từ: “key” =value, gọi ra: <tên Obj>[key]

or <tên Obj>. [biến]

Kiểm tra sự tồn tại của thuộc tính trong Obj: “key” in <tên Obj>

Vòng lặp Obj:

for (let <key> in <tên Obj> {

key //lấy ra key

<tên Obj>[key] //lấy giá trị

}

* ***Tham chiếu đối tượng và sao chép***

- 1 biến đc gán cho 1 đối tượng: là lưu trữ địa chỉ của đối tượng trong bộ nhớ, nói cách khác là 1 tham chiếu tới nó.

- Khi 1 biến đối tượng đc sao chép, tham chiếu sẽ đc sao chép, nhưng bản thân đối tượng thì k.

* ***Nhân bản, hợp nhất đối tượng***

Object.assign(dest, [src1, src2, …]) hợp nhất vào dest

Dest: đối tượng đích

Src: các đối tượng nguồn

* ***Object method “this”***

- Các hàm đc lưu trữ tr thuộc tính của đối tượng đc gọi là phương thức

- method shorthand: bỏ function

user = {

sayHi: function() {

alert("Hello");

}};

user = { //shorthand

sayHi() {

alert("Hello");

}};

**-**  Để truy cập đối tượng, 1 phương thức có thể sd từ khóa “this”

* ***Function Constructor, operator “new”***

function <Tên hàm>(biến){code}

let <tên Obj> = new Object ();

**-** Quy ước: đặt tên viết hoa chữ cái đầu, thực thi vs toán tử “new”

- Các bước:

1 Obj mới đc tạo và gán this cho đối tượng này

Thân hàm đc thực thi

Giá trị của “this” đc trả về

**10. Object khác**

* ***Array***

- Là 1 loại Obj đặc biệt, lưu trữ và quản lý các mục dl có thứ tự

- 2 syntaxes:

Let <tên Arr> =new Array ();

Let <tên Arr> = [];

+ ***Thêm, sửa, xóa***

- push(): thêm 1 or n phần tử vào cuối mảng, return độ dài mới của mảng

- pop(): xóa 1 phẩn tử cuối mảng và in ra phần tử đã xóa

- shift(): xóa 1 phẩn tử đầu mảng và in ra phần tử đã xóa

- unshift(): thêm 1 or n phần tử vào đầu mảng, return độ dài mới của mảng

- splice: chèn, xóa, thay thế phần tử, trả về mảng các phần tử bị loại bỏ

Splice (vị trí bắt đầu, số element muốn xóa, chèn element mới)

- slice (start, end): cắt 1 or n element từ mảng, trả về mảng mới sao chép

+ ***Nối Array***

- concat(): nối arraay

- toString(): return chuỗi, phân cách bằng dấu ,

+ ***Loop***

- vòng lặp: for..of: duyệt mảng

forEach(function(item, index, array)): chạy 1 hàm cho mọi phần tử mảng

+ ***Search***

-indexOf(item, from), lastIndexOf(), includes()

- find(function(item, index, array)): tìm 1 phần tử đầu vs điều kiện cụ thể

findIndex: return chỉ mục nơi phần tử đc tìm thấy

- filter(fn): tìm tất cả các phần tử vs đk tìm kiếm, return Aray

+ ***Chuyển đổi 1 Array***

- map(fn): sửa và lấy ra element của 1 or n phần tử, return Array

- sort(fn(a,b)): sắp xếp mảng tại chỗ

- reverse(): đảo ngược chuỗi

- split(‘điểm chung’, độ dài mảng-tùy chọn): tách chuỗi thành mảng

- join(): nối các phần tử mảng, return chuỗi

- reduce(): trả về 1 giá trị duy nhất sau khi tính toán

Reduce(function(biến lưu trữ, item, index, array){}, gtri khởi tạo)

- Aray.isArray(biến ktra): kiểm tra 1 mảng có phải array k

* ***Map and Set***

- Map: tập hợp các mục dl có key (giống Obj), khóa thuộc bất kỳ loại nào

- Set: tập hợp các giá trị (k có khóa), mỗi giá trị chỉ xuất hiện 1 lần

* ***Date and time***: lưu thông tin ngày giờ

-Tạo Obj Date: new Date()

* ***Error***: lưu thông tin về 1 lỗi

**11. JSON methods** (JS Obj Notation)

- là 1 định dạng dữ liệu (chuỗi)

- thể hiện dạng dl: Number, Boolean, Null, Array, Object

JSON.stringify(dl chuyển đổi): chuyển từ JS sang Json

JSON.parse(): chuyển từ Json sang JS

**12. Prototypes, Inheritance**

- Các Obj có 1 thuộc tính ẩn: [[Prototype]] có giá trị rỗng or tham chiếu đến 1 đối tượng khác (nguyên mẫu)

\_\_proto\_\_: kế thừa từ 1 Obj, là getter, setter cho [[Prototype]]

For..in: lặp qua tất cả thuộc tính, cả thuộc tính kế thừa

- Thiết lập [[Prototype]] cho các đối tượng đc tạo thông qua hàm tạo

F.prototype đặt các đối tượng mới khi new F() đc gọi, giá trị là Obj or null

- Cách mới:

Object.create(proto, [desc]): tạo 1 Obj trống kế thừa proto,

desc: (thuộc tính bổ sung)

Object.getPrototypeOf(obj)

Object.setPrototypeOf(obj, proto)

**13. Classes**

**-** Là khuôn mẫu để tạo ra đối tượng sd “new” (là 1 loại hàm)

Class <tên class> {

Prop=value; //property

Constructor () {}

Method1() {}

Method2() {} …

get something(){}

set something(){} …

}

* ***Class kế thừa***: extends

-Override method:

super.method(…): gọi 1 phương thức cha

super(…): gọi hàm tạo cha

* ***Static properties and method***

- Gán 1 phương thức cho chính class chứ k phải gán cho ‘prototype’: sd static

- static:sd cho chức năng thuộc về lớp, khi muốn lưu trữ dl mức lớp

static property=…;

static method(){…}

* ***protected, private properties and method***

- Trong OOP: thuộc tính và phương thức đc chia thành 2 nhóm:

+ internal interface: k thể truy cập từ bên ngoài

Sd các thuộc tính

Pretected: start with \_ chỉ truy cập đc bên trong lớp và các lớp kế thừa

Private: start with # chỉ truy cập đc bên trong lớp

+ external interface: có thể truy cập từ bên ngoài

* ***Instanceof***

- Kiểm tra 1 đối tượng có thuộc 1 lớp nào đó k (or kế thừa k), return boolean

Obj instanceof Class

* ***Mixins***

- Là 1 lớp chứa các phương thức có thể đc sd bởi các lớp khác mà k cần kế thừa từ nó

- Triển khai 1 mixin là tạo 1 obj vs các phương thức để hợp nhất thành 1 prototype của bất kỳ class nào.

**14. Promise, async/await**

* ***Callback:*** là hàm, đc truyền qua đối số, đc gọi lại trog hàm nhận đối số, hàm ngoài chạy xong thì hàm trong ms đc chạy
* ***Promise***

+ sync: đồng bộ, viết trước chạy trước

+ async: bất đồng bộ

Vd: setTimeout, setInterval, fetch, XMLHttpRequest, file reading, request animation frame

-> Callback để giải quyết Async

+ Nhiều callback lồng nhau, code rối -> sinh ra vấn đề: callback hell

***-> Promise (ES6)*** sinh ra: sử lý các thao tác bất đồng bộ, khắc phục tình trạng callback hell.

***1. Khởi tạo Promise***

var <promise> = new Promise (

//2. Executer: trả về 2 tham số là hàm

Function (resolve, reject) {

//logic

//thành công: resolve ()

//thất bại: reject ()

}

);

* ***Sử dụng:***

<promise>

// Nếu resolve đc gọi thì callback của then sẽ đc gọi, return cái gì thì then phía sau sẽ nhận đc cái đó

.then(function(){…})

.then …

//nếu reject đc gọi thì callback của catch sẽ đc gọi

.catch(function(){…})

//1 trong 2 đc gọi thì finally đều đc gọi

.finally(function(){…})

* ***3 trạng thái của Promise***

- Pendding: k resove, k reject -> rò rỉ bộ nhớ

- Fulfilled: thành công

- Rejected: thất bại

* ***Cách tạo Promise khác***

- Tạo 1 promise thành công: var promise = Promise.resolve(value);

- Tạo 1 promise thất bại: var promise = Promise.reject(error);

- Chạy song song các promise: Promise.All(); có 1 promise reject thì sẽ reject

//nhận đối số là 1 mảng, return promise

Promise.All([promise1, promise2, …])

//khi tất cả promise trên song thì ms lọt vào .then

.then(function(result){…}); //trả về 1 mảng

**15. Modules**

- Ứng dụng lớn đc chia thành nhiều tệp, mỗi tệp đgl 1 module

- 1 module chứa 1 lớp or 1 thư viện các chức năng cho 1 mục đích cụ thể

- Các moduls sd export, import để trao đổi chức năg

* ***Tính năng cốt lõi***

- Luôn hoạt động ở chế độ nghiêm ngặt (stric)

- Phạm vi cấp module: các biến và hàm trog 1 module k đc thấy trog các tập lệnh khác

- Mã module chỉ đc đánh giá lần đầu tiên khi đc nhập

import.meta: chứa thông tin về trang web hiện tại

trog 1 module, “this” là undefined

* ***Tính năng cho brower***

- Để nhập, xuất, trình duyệt cần <script type=”module”>: cho browers biết tập lệnh phải đc coi như 1 module

+ Đc hoãn lại theo mặc định

+ Async làm việc trên inline script

+ Để tải external script từ nguồn khác, tiêu đề CORS là cần thiết

+ external script trùng lặp bị bỏ qua

**16. BOM**

Brower Object Model

- Đại diện cho các đối tượng bổ sung đc cung cấp bởi trình duyệt để làm việc vs mọi thứ ngoại trừ tài liệu

**17. DOM**

Document object Model: là tiêu chuẩn của w3c

- Khi 1 trang web đc tải, brower sẽ tạo ra 1 DOM của trang.

- Html DOM đc xây dựng như 1 cây các đối tượng gồm: element nodes (tags), attribute nodes, text nodes ->dùng để lấy, thêm, sửa, xóa các phần tử html

- Với 1 node: có thể lấy ra các nút lân cận

+ All nodes: parentNode, childNodes, firstChild, lastChild, previousSiblings, nextSibling

+ element node only: parentElement, children, firstElementChild, lastElementChild, previousElementSiblings, nextElementSibling

* ***Searching nodes***

querySelector, querySelectorAll, getElementById, getElementByName, getElementByTagName, getElementByClassName.

- elem.matches(css): check elem có khớp với css selector k

- elem.closest(css): tìm tổ tiên gần nhất có khớp vs css selector k, check cả elem.

- elemA.contain(elemB): return true nếu elemB nằm trog elemA or bằng nhau

* ***Node properties***

- nodeType: ktra 1 node là 1 text or 1 element node

- nodeName/tagName: đọc tên thẻ

- innerHtml: lấy or thêm elem, attribute, text vào trog element

- outerHtml: lấy từ elem đc gọi

- nodeValue/data:

- textContent: nd text lấy đc là những text node trog element

- hidden: true (tương tự display: none)

* ***Atribute and properties***

- attribute: viết bằng html, k pb hoa thường, type: string

- properties: viết trong đối tượng DOM, pb hoa thường, type: any

- Method work with attribute:

+ elem.hasAttribute(name): check sự tồn tại

+ elem.getAttribute: lấy value

+ elem.setAttribute(name, value): đặt value

+ elem.removeAttribute(name): xóa attribute

+ elem.attributes: tập hợp tất cả attribute

Chỉ tham chiếu đến attribute khi property DOM k phù hợp

* ***Sửa đổi tài liệu***

+ Create new nodes

document.createElement(tag): tạo 1 elem vs thẻ đã cho

document.createTextNode(value)

elem.cloneNode(deep): sao chép elem nếu deep==true vs all các phần tử con

+ Chèn và xóa …

* ***Styles and classes***

+ Để quản lý DOM properties:

- className: giá trị string, tốt để quản lý toàn bộ class

- classList: return DOMTKenList, manage class of element:

elm.classList.length: số class trog elem

elem.classList.add(class): thêm class vào elem

elem.classList.contains(class): check class có tồn tại tr elem k

elem.classList.remove(class): xóa class

elem.classList.toggle(class): có class thì xóa, k có thì thêm

+ set styles

elem.style.thuộc tính; //1 style

Object.assign(elem.style, {css…}; //nhiều style

+ get styles: elem.style.thuộc tính;

* ***Element size and scrolling***

offsetParent: tổ tiên gần nhất

offsetLeft/offsetTop: tọa độ vs cạnh trên bên trái của offsetParent

offsetWidth/offsetHeight: width, height bên ngoài của elem, cả đg viền

clientLeft/clientTop:

clientWidth/clientHeight: width, height of content, k có scrollbar

scrollWidth/scrollHeight: width, height of content, có scrollbar

scrollLeft/scrollTop: width, height of phần đc cuộn tính từ góc trên bên trái

**18. Brower events**

3 cách để chỉ định trình xử lý sự kiện

- Html attribute: onclick= “…”, ít sd

- Dom property: elem.onclick=function

- Method: elem.addEventListener(even, handle[, phase]) to add

romoveEventListener to remove

**19. Bubbling and capturing**

* ***Bubbling: nổi bọt***

- Khi 1 sự kiện sảy ra trên 1 element, trước tiên nó chạy các trình xử lý trên nó, sau đó chạy trên cha của nó, sau đó chạy lên tổ tiên khác

* ***Even.target***

-Phần tử đc lồng sâu nhất đã gây ra sự kiện là phần tử đích, có thể truy cập dưới dạng event.target

* ***Stopping bubbling***

event.stopPropagation()

**20. Brower default actions**

mousedown: kích chuột xuống

click: on input checkbox

submit: in <input>

keydown: nhấn 1 phím

contextmenu: nhấp chuột phải, hiển thị menu trình duyệt

* ***prevent default action:***

event.preventDefault() or return false

**Bổ xung**

**1. IIFE:** Immediately Invoked Function Expression

- Là function expression đc gọi ngay lập tức

(function(){code})()

+ dùng dấu; trước IIFE

+ là hàm private: có tính đóng gói, thông qua các phương thức public để truy cập

**2. Scope:** phạm vi

- global: toàn cầu

- code block: khối mã{} (let, const)

- local scope: hàm (var, function)

+ Khi gọi mỗi hàm luôn có 1 phạm vi mới đc tạo ra

+ Các hàm có thể truy cập các biến đc khai báo trong phạm vi của nó và bên ngoài nó

**3. Closure**

- Là hàm có thể ghi nhớ nơi nó đc tạo ra và truy cập đc biến ở bên ngoài phạm vi của nó

- Ứng dụng private trong OOP

- Biến đc tham chiếu trong closure sẽ k đc xóa khỏi bộ nhớ khi hàm cha thực thi xog.

**4. Hoisting**

- Đưa khai báo biến lên trên đầu phạm vi của chúng

+ Với var, function declaration (function tênF)

hoist và khởi tạo giá trị là undefined

+ Với let, const:

hoist nhưng k đc tạo giá trị và đc đưa vào ‘Template Dead hoisted’

**5. Strict mode:** chế độ nghiêm ngặt

+ Cách sd:

- Thêm ‘use strict’: vào đầu file js, sau thẻ mở <script>, or đầu phạm vi hàm.

+ Đặc trưng:

- Báo lỗi khi gán lại giá trị cho thuộc tính có writable: false (k đc sửa)

- Báo lỗi khi hàm có tham số trùng tên

- Khai báo hàm trog code block thì sẽ thuộc phạm vi của code block

- k đặt tên hàm, tên biến bằng từ khóa

**6. Value types & Reference types**

* ***Value types:*** tham trị

- Là kiểu dl nguyên thủy: string, number, Boolean, bigInt, symbol, undefined, null

let a=1; Khi 1 biến đc tạo, cấp 1 ô nhớ, lưu giá trị vào ô nhớ

* ***Reference types:*** tham chiếu

- Là object, array, function

const a= {…}; Khi 1 biến đc tạo, cấp 1 ô nhớ, lưu obj vào ô nhớ, trả về địa chỉ đã lưu và gán cho biến a.

- Mỗi Obj đc lưu trữ tại 1 vùng nhớ khác nhau

- 1 biến đc gán cho 1 đối tượng: là lưu trữ địa chỉ của đối tượng trong bộ nhớ, nói cách khác là 1 tham chiếu tới nó.

- Khi 1 biến đối tượng đc sao chép, tham chiếu sẽ đc sao chép, nhưng bản thân đối tượng thì k