

Shortcut

- `header && header.classList.add("is-active"); ==> if(header){ header.classList.add("is-active") }`
- `document.DocumentElement(thẻ)`

##Chữ radiant `-webkit-background-clip: text; -webkit-text-fill-color: transparent;`

-các template được tạo ra trong DOM vì được sinh ra sau khi DOM đã load xong, khi web load xong rồi DOM add sự kiện không có nên trả về null -> Thay vào đó ta bắt sự kiện từ body

-- Khi dùng `setItem` => giá trị truyền vào phải là string => dùng `JSON.stringify` => Ex:
`localStorage.setItem('todo_list',JSON.stringify(todoList));`

- Cần khai báo biến trước khi gọi biến đó
- Let và const đều bị hoisting
- Var không bị hoisting có thể gọi biến var trước khi khai báo nó
- Function con có thể truy xuất tới scope của function cha.

-Ex1: `function sayHello(){ let message = "hi"; function sayHi(){ console.log(message); }`

```
return sayHi;
```

```
}
```

`const hello = sayHello()` //Lúc này hàm `sayHi` chưa được gọi vì return về một hàm `hello()` // Thế là hàm `sayHi` được được chạy

-Ex2: `function sayHello3(message){ return function sayYourName(name){ console.log(`${message} ${name}`); } }`

`const hello2 = sayHello3("Hi I am"); hello2("Clousure");`

Date

- Lấy timestamp ta dùng hàm `getTime()`;
- timestamp được tính bằng `milisecond`
- `new Date(year,month,day,hour,minutes,second,milisecond)`
- `new Date(timestamp)`
- `new Date(date string)`;
- Tháng trong date bắt đầu từ 0-11
- Thứ trong tuần từ 0 -> 6. 0 -> Chủ nhật, 6 -> Thứ 7
- Tháng là phải trừ đi 1
- Để lấy ngày tháng năm của VN vd: `const now = new Date(); now.toLocaleString("vi-VI");` //In ra ngay tháng năm của VN

String

Slice

- Hàm `slice(startIndex, endIndex - 1)`;
- Lấy từ phần tử `startIndex` -> `endIndex - 1`

Splice

-Xoá hoặc thay thế phần tử gốc làm thay đổi mảng gốc

- `splice(startIndex);` => Tương tự như `slice(startIndex)`;
- `splice(startIdx, countDelete);`
- `splice(startIdx, countDelete, item1, item2, itemN)`; //item1, item2, itemN là những giá trị sẽ thay thế



Sort

=> Sắp xếp các giá trị theo bảng mã Unicode-16 function `sort((a,b)=>{ if(a>b) return 1; //Tăng dần if(a<b) return -1; // Giảm dần })`

Sự khác nhau giữa `forEach` và `map`

`ForEach` thì không return, thường dùng chạy trong DOM, không thể dùng

`Map` thì trả về một mảng mới dựa vào mảng ban đầu

Sự giống nhau giữa `some` và `every` (đều trả về boolean)

Để so sánh 2 mảng [array] ta dùng `JSON.stringify` để so sánh 2 mảng đó



2 cách sao chép mảng (clone) : dùng `slice` & `spread operator`:
`[...tenmang]`

2 cách gộp mảng : dùng `concat` && `spread operator`
`[...mang1,...mang2,...mang3]`

Destructuring [x,y,z] = name array => tương ứng với từng giá trị trong array

Để làm phẳng mảng ta dùng flat(số mảng con)

-> vd: flat(1) -> làm phẳng 1 mảng con



Các giá trị falsy là : 0,undefined,null,NaN,"",false;

Number

-> Math.sign(number) -> number là số dương => = 1, nếu là số âm => = -1;

- VD: Math.sign(12) // 1
- VD: Math.sign(-12) // -1

[Object]

```
const student = {name:'Tong Nguyen',age:22,love:true,isDeveloper:true,hi: function(){console.log("Hi")}}
```

Để xoá key trong object ta dùng phương thức delete

-> delete student.name;

Object.keys: trả về một mảng chứa các keys của object

Object.entries: trả về một mảng nested array [["name"], ["Tong Nguyen"]] gồm có các cặp key và value



Object.assign() dùng để gộp nhiều object lại với nhau



Object.freeze(object) dùng để đóng băng key và value của object không thể thêm key và value mới vào object đã đóng băng



`Object.seal(object)` cho phép chỉnh sửa key & value nhưng không được thêm key & value mới



Để sao chép một object dùng spread operator `{...object}` hoặc `Object.assign()`

Để sao chép một object clone nested ta dùng `JSON.parse(JSON.stringify(tên object))` thì object ta có thể thay đổi thuộc tính mà không ảnh hưởng tới object gốc



Optional chaining (?) khi ta muốn truy xuất nhiều keys lồng nhau khi chưa khai báo ta dùng nó để hạn chế gây lỗi khi chưa có key của object đó



!Lưu ý: Thông thường `typeof` của `[]`, `{}`, `null` là object nên muốn kiểm tra có phải là object không thì nên xét hết các điều kiện

Để kiểm tra một object có chứa keys nào đó hay không ta dùng `(object.hasOwnProperty(key))` nếu có trả về `true`, còn không có thì trả về `false`

rest parameter (...rest) khi nó trong một function thì là một array

vd: `function sum(num1,num2,...rest){ => num1 = 1 => num2 = 2 => rest = [3,4,5,6] } sum(1,2,3,4,5,6)`

`querySelectorAll`: nó có thể dùng loop và chỉ sử dụng được `forEach` !!

=> Sẽ trả về một `NodeList` giống như array

attribute -> thuộc tính : `href,class,id,src,style,...`

- `selector.getAttribute("attribute")` => Lấy ra giá trị của attribute selector.
- `selector.removeAttribute("attribute")` ==> Xóa attribute selector
- `selector.hasAttribute("attribute")` => Kiểm tra xem có attribute hay không, nếu có -> `true`, không -> `false`

Node

Kiểm tra element có childNode hay không, có -> true, không -> false

=> element.hasChildNodes()

insertAdjacentText, insertAdjacentHTML, insertAdjacentElement

element.insertAdjacentText("position", "string");

- position: beforebegin, afterbegin, beforeend, afterend
- string: text

 ScreenShot

Để xoá Node ra khỏi DOM ta dùng removeChild => khi muốn xoá một nốt con ta tìm lên phần tử cha rồi mới xoá được node đó

- vd: selector.parentNode.removeChild(selector);

Để lấy được element cha ta dùng

- parentNode
- parentElement

nextElementSibling chọn phần tử kế tiếp nó

vd: span.nextElementSibling; => element span2 span.previousElementSibling => element p

[a](#)

previousElementSibling chọn phần tử trước nó

childNodes: trả về 1 mảng hết các node bên trong bao gồm textNodes

children: trả về 1 mảng hết các node bên trong không bao gồm textNodes



firstChild : trả về element đầu tiên bao gồm textNodes

firstElementChild : trả về element đầu tiên không bao gồm textNodes

lastChild & lastElementChild tương tự như trên nhưng lấy element cuối cùng

Offset sizing && client

Offset

- element.offsetWidth => trả về width của element đó
- element.offsetHeight => trả về height của element đó
- element.offsetLeft => vị trí của nó so với bên trái
- element.offsetTop => vị trí của nó so với phía trên
- element.offsetParent => lấy ra phần tử cha để lấy giá trị của phần tử cha  

Client

- element.clientWidth => độ rộng của phần tử trừ đi border (2 bên)
- element.clientHeight => chiều cao của phần tử trừ đi border (trên dưới) (viewport)
- element.clientLeft => vị trí của nó so với bên trái border (!lấy border-left:)
- element.clientTop => vị trí của nó so với bên trên border (!lấy border-top:)

 Screenshot

For...in trong mảng thì lấy ra index của từng phần tử trong mảng

window.innerWidth, window.outerWidth, window.innerHeight, window.outerHeight

- inner là lấy vị trí của khung nhỏ (width | height)
- outer là lấy toàn khung màn hình (width | height)

 Screenshot

selector.getBoundingClientRect() -> Lấy ra kích thước toạ độ của phần tử

- left,x: lấy vị trí của khối so với bên trái
- top: vị trí của khối so với bên trên
- bottom: chiều cao của khối + top
- right: độ rộng của khối + left
- width: độ rộng của khối
- height: chiều cao

HTMLCollection && NodeList

- Điểm giống: có thể truy cập bằng index, có độ dài length, giống mảng nhưng không hẳn là mảng tức không thể sử dụng các phương thức của mảng như push,map,shift,filter
- HTMLCollection: không thể sử dụng forEach
- NodeList : Sử dụng được forEach

Có 2 cách convert HTMLCollection && NodeList (phiên bản thấp) to array => sau đó nó có thể dùng forEach, filter,...

-> Array.from(HTMLCollection,NodeList) -> [...HTMLCollection,...NodeList] => operator

ParentNode && ParentElement

- parentElement: có thể null
- parentNode: nó lấy thể hiện tại của nó nếu nó không có cha lớn nhất bao ngoài

insertBefore

-> parentNode.insertBefore(newnode,existingnode) => đưa element newnode lên trên element existingnode đã tồn tại.

replaceChild(newnode,oldnode)

-> node.replaceChild(newnode,oldnode)

Lưu ý

- Lỗi sai khi dùng function trong eventlistener

//Sai vì khi chưa click vào mà function đã involk nên dẫn đến sai sự kiện

element.addEventListener('click',handleClick()) // Đúng element.addEventListener('click',handleClick);

Bubbling

- Bắt sự kiện từ trong ra ngoài (từ thẻ click đến thẻ bên ngoài) // Lúc này chỉ có span chạy vì đã chặn sự nổi bọt lên các phần tử cha // `e.stopPropagation()`; // Nó chỉ chặn 1 sự kiện click, nếu phần tử có thêm một sự kiện thì nó sẽ chạy 2 sự kiện muốn chặn 1 sự kiện ta dùng `stopImmediatePropagation()`;

Capturing

- Bắt sự kiện từ ngoài vào trong
- Chuyển vào parameter là một object có key(capture: true) default: false; nếu không truyền thì nó sẽ bị bubbling // `span.addEventListener("click", function(e) { console.log("span click 2"); }, { capture: true, });`

 Screenshot

h3 Để random các phần tử trong một mảng ta dùng `arr[Math.floor(Math.random() * arr.length)]`

`event.target` như một element của nó = this trong function (phần tử cha)

h2 Toggle của classList: khi một class đã có rồi thì click vào nó sẽ xóa và khi nhấn một lần nữa thì nó sẽ được thêm vào

h2 `selector.contains()`: kiểm tra element hoặc className có chứa element khác hay không

`e.target.matches(".className")`: kiểm tra có trùng với className hay không

Event mouse

- `mousemove`: khi di chuyển chuột trên phần tử đó
- `mouseover`: nó sẽ chạy khi rê chuột vào phần tử hoặc con của phần tử đó (từ trong đó rơi vào thẻ con của nó vẫn tính là 1 lần)
- `mouseenter`: nó sẽ chạy khi rê chuột vào phần tử
- `mouseleave`: nó sẽ chạy khi rê chuột ra khỏi phần tử
- `wheel`: lăn chuột (có `e.delta` => lăn lên => dương, lăn xuống => âm)

ClientXY và PageYY lấy toạ độ khi rê chuột

- clientY: Lấy chiều cao theo viewport
- PageY: lấy chiều cao theo document (thường thì có scroll nó sẽ khác viewport)

Form event

- keydown: Khi nhấn xuống (có preventDefault) (sẽ lấy hết các phím)
- keyup: Sự kiện xảy ra khi nhấn phím rồi thả ra (không the dùng preventDefault) -keypress: sẽ ignore(không nhận) các phím như home,pgUp,pgDown,Esc,Shift,Delele,Alt,Ctrl,.. (use preventDefault) -> thứ tự ưu tiên keydown -> keypress -> keyup -change: khi giá trị thay đổi và click ra ngoài hoặc nhấn enter trong 1 lần
- focus: khi focus vào ô input sự kiện mới xảy ra
- blur : khi blur click ra khỏi ô input sự kiện mới xảy ra
- input: lấy giá trị khi chúng ta gõ

Scroll

- debounce: là một kĩ thuật buộc một hàm phải đợi một khoảng thời gian nhất định trước khi thực thi. Trong khoảng thời gian đợi, một tác động sẽ điều bị bỏ qua và chỉ nhận duy nhất 1 hành động diễn ra trong thời gian chúng ta định trước.
- window.pageYOffset: khoảng cách thanh scroll của window theo chiều dọc so với phiếu trên cùng.
- window.pageXOffset: khoảng cách thanh scroll của window theo chiều ngang so với phiếu bên trái - scrollHeight: trả về chiều cao của element bao gồm paddig (+padding 2 bên), nhưng không có border - scrollWidth: trả về chiều rộng của element bao gồm paddig (+padding 2 bên), nhưng không có border
 - element.scrollTop: trả về thanh scroll so với trên cùng (windown không dung duoc) (tựa như pageYOffset)
 - element.scrollLeft: trả về thanh scroll so với bên trái (windown không dung duoc) (tựa như pageXOffset)
- element.scrollIntoView() : scroll tới phạm vi mà có thể thấy nó (có thể quay lại phần tử đƯợc trỏ tới)

-> <https://s1.o7planning.com/vi/12397/images/41666223.png>

offsetWidth & scrollWidth (offsetHeight & scrollHeight tương tự)

- offsetWidth -> Lấy ra độ rộng khi chúng ta set nó ban đầu
- scrollWidth -> Lấy ra độ rộng khi xuất hiện scroll hoặc có phần tử con lớn hơn độ rộng của nó.

Để tính độ dài thực sự của scroll: ta lấy chiều cao của scroll(scrollHeight / scrollWidth) - chiều cao của viewport height (clienHeight or offsetHeight / clientWidth or offsetWidth)

Cách lấy các thẻ input nhanh bằng cách gọi form.elements["name của input"]
