🏠_Home_

🔗https://app.diagrams.net/#G1rT-Vton5_5Qdp_znlz1Nb1eMP_zA1B9I
🔗 JavaScript(Sololearn)
🔗 JavaScript Cheatsheet.pdf
🔗[That Weird JavaScript Course - Fireship]()

JavaScript

- Usage → Static page එකක් dynamically update කරන්න

What is javascript ?

> **JavaScript** (**JS**) is a lightweight, interpreted or JIT compiled programming language with first-class functions. Most well-known as the scripting language for Web pages, many non-browser environments also use it, such as node.js and Apache CouchDB. JS is a prototype-based,Loosly Typed, dynamic scripting language, supporting object-oriented, functional programming styles.

- Lightweight , interpreted , JIT compiled language එකක් first class functions තියෙන.
- ඊට අමතරව prototype based , loosely typed, OOP support, dynamic scripting language එකක්

- Browser, non-browser environment 2ක run කරන්න පුලුවන් (node.js ← non browser based)
- Browser එකෙ run වෙන නිසා scripting language එකක්
- මුලින් scripting language එකක් විදියට introduce කලේ, පස්සෙ programming language එකක්

History

Brendan Eich(1995) → Mocha (LiveScript) → JavaScript → ECMAScript 1 (1997) → ECMAScript 2. . . ECMAScript 13 (ES2022)

Javascript standard versions → ECMAScript (ECMA community එක හරහා manage වෙන්නෙ)

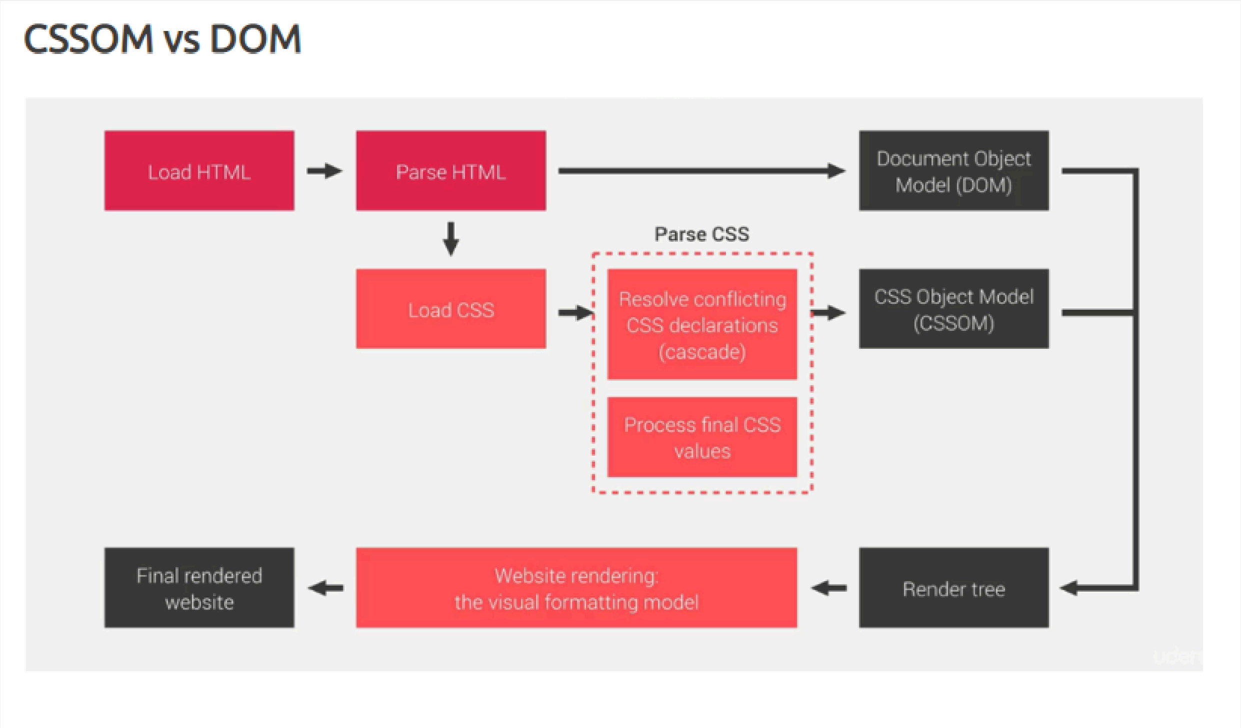
Java vs JavaScript

දෙකම Syntactically සමානයි, OOP support

Java	**JavaScript**
Class-based	Prototype-based
Object එකකට dynamically properties add කරන්න බෑ	පුළුවන්
Variable declaration එකේදි type එක කියනතෝ (Strong/static type Language එකක්)	එහෙම කියන්න ඕන නෑ (loosely type language එකක්)
Hard එකෙ auto write කරන්න පුළුවන්	බෑ

Browser එකෙ HTML file එකක් process කරන විදිය

1. HTML Parsing
2. Document Object Model (DOM) Construction
3. CSS Parsing and Style Computation
4. Layout (or Reflow)
5. Painting
6. rendering and Display



Javascript, DOM manipulate කරන්න use කරයි. (= web page එක වෙනස් කරන්න)

Integrate JS with HTML

ක්‍රම 2 කට පුලුවන්,

1. Internal script
2. External script

```
<!DOCTYPE html>
<html lang="en">
<head>
  <script>

  </script>
</head>

<body>

<!--Internal Script-->
<script>
    alert("Hello there How are you.?");
```

```html
</script>

<!--External Script-->
<script src="assests/js/myScript.js"></script>
</body>

</html>
```

<script> tag එක යටින් දාන්න හේතුව

Javascript, DOM (Document Object Model) manipulate කරයි.
Dom එක load වෙලා ඉවර වෙන්නෙ body එකෙ closing tag එකනේ පස්සෙ,
ඒ නිසා **DOM එක full create වුනාට පස්සෙ script එක වෙන්නෙ**, script tag එක body එකෙ closing tag එක ලගින් දානො.

(උඩින් දැම්මත් වැඩ/සමහර වෙලාවට නෑ, යටින් දාන එක වඩා සුදුසු "Placing scripts at the bottom of the <body> element improves the display speed, because script interpretation slows down the display.") 🔗 [JavaScript Where To]

Output

1. Alert

```javascript
alert("Hello");
```

2. Console output
   - Normal

```javascript
console.log("Hello");
```

   - Warning

```javascript
console.warn("Warning!");
```

   - Error

```javascript
console.error("ERROR!");
```

3. using viewport

select elements from the DOM

access the DOM object → document.

Select single element

```
document.getElementById("btn1");
```

Can select an item using an element name attribute // NodeList

```
document.getElementsByName("txtContent");
```

can select an element using a class name // HTMLCollection(3)

```
document.getElementsByClassName("btn");
```

can select an element using an element name// HTMLCollection(2)

```
document.getElementsByTagName("h1");
```

can use this one to select DOM elements using CSS selectors

```
document.querySelector("h1");
```

Remove selected

```
document.getElementsByTagName('h1')[0].remove();
```

Variables

 # JS Variable declaration types var, let, const

Loose type
JavaScript වල Variable එකක් declare කරද්දි type එකක් දාන්නෙ නෑ. ඒ නිසා JS loose type.

Dynamic type

Type එක dynamically වෙනස් වෙනෝ assign කරන value එක අනුව.

* (var, let වැඩ . const වලට වැඩ නෑ.)

```
> var address = "galle";
<· undefined
> typeof address;
<· 'string'
> address = 10;
<· 10
> typeof address;
<· 'number'
```

Get variable type :

```
typeof variableName;
```

var

- variable name එකම duplicate කරන්න පුලුවන්. ගන්නෙ අන්තිමට තියෙන එක.

```
> var name = "amal";
  var name = "kamal";
  console.log(name);
  kamal
```

- Scope එක ගැන බලන්නෙ නෑ

```
{
    var z = 10;
}

console.log(z);
10
```

- Variable එකක් declare කරන්න කලින් use කරන්න පුලුවන් error එකක් නැතුව (**Hoisting**), initialize වෙන්නෙ පස්සෙ හැබැයි.

```
> console.log(y);
  var y=10;
  console.log(y);
  undefined
  10
```

let

var එකෙ තිබුණ flexibility එක නැ.
Dynamic typing, loose typing වැඩ
Duplicate, Out of Scope, Hoisting වැඩ නැ.

- Let එක දාල හදපු variable duplicate කරන්න බැ

```
let name = "x";
let name = "y";
Uncaught SyntaxError: Identifier 'name' has already been declared
```

- Scope ගැන බලනෝ.

```
{
    let a = 10;
}
console.log(a);
Uncaught ReferenceError: a is not defined
    at <anonymous>:4:13
```

- Hoisting support නැ.

```
console.log(y);
y=20;
console.log(y);
Uncaught ReferenceError: y is not defined
    at <anonymous>:1:13
```

const

```
const v ;
Uncaught SyntaxError: Missing initializer in const declaration
> const v = "value" ;
  console.log(v);
  value
```

Constants.
Declare කරද්දිම value එකක් දෙන්න ඕන.
loos type වැඩ. Scope එක ගැන බලනො.(let වගේ)

	var	let	const
Loose type	✔	✔	✔

Dynamic type	✔	✔	❌
Variable name duplicate	✔	❌	❌
Scope	❌	✔	✔
Hoisting support	✔	❌	❌

====

JS case sensitive.

Java වල flow controls එහෙමේමම තියෙනෙො,

🔗 JavaScript(Sololearn)

====

- **if, if else, switch case, ternary operator** java වල විදියමයි.

Loops

- For Loop
- While Loop
- Do While Loop
- For Each
- For In
- For Of

- **for, while, do-while** java වල විදියමයි.

Special loops for arrays

```
var names=["Amal","Kamal","Wilam","Namal"];
```

- **for-each loop**

  Array එකෙ last value එක වෙනකල් for each function එක ආයෙ ආයෙ call වෙනෙො.

```
names.forEach(function (v){
    console.log(v);
});
```

Output

```
Amal
Kamal
Wilam
Namal
```

- **for-in loop**

  can get the **index numbers** of an array

```
for(var i in names){
    console.log( i,names[i]);// i refer index of the array
}
```

Output

```
0 Amal
1 Kamal
2 Wilam
3 Namal
```

- **for-of loop**

  can get the values of the array

```
for(var i of names){
    console.log(i);// i refer values of the array
}
```

Output

```
Amal
Kamal
Wilam
Namal
```

Arrays

- literal base array

```
let colors = ["red","green","orange"];
```

- function base array or constructor base array

```
let Colors = new Array("red","green","orange");
```

Array related methods

- **arr.pop();**

  last index එක remove කරනො.

```
> arr
<· ▸(4) ['A', 'B', 'C', 'D']
> arr.pop();
<· 'D'
> arr
<· ▸(3) ['A', 'B', 'C']
```

- **arr.push("New Value");**

  array එකේ last index එක ලගට අලුත් element එකක් add කරනො.

```
> arr
<· ▸(3) ['A', 'B', 'C']
> arr.push("K");
<· 4
> arr
<· ▸(4) ['A', 'B', 'C', 'K']
```

- **arr.shift();**

  array එකෙ 1st index එක remove කරනො.

```
> arr
<· ▸(5) ['A', 'B', 'C', 'D', 'E']
> arr.shift();
<· 'A'
> arr
<· ▸(4) ['B', 'C', 'D', 'E']
```

- **arr.unshift("New Value");**

  array එකෙ 1st index එක ලගට අලුත් element එකක් add කරනො.

```
> arr
<- ▸(5) ['A', 'B', 'C', 'D', 'E']
> arr.unshift("X");
<- 6
> arr
<- ▸(6) ['X', 'A', 'B', 'C', 'D', 'E']
```

- **arr.slice(start_index , end_index);**

  Index number range එකක් දීල,
  array එකකින් කෑල්ලක් වෙන්කරගෙන අලුත් array එකක් හදන්න.

```
> arr
<- ▸(6) ['X', 'A', 'B', 'C', 'D', 'E']
> arr.slice(2,4);
<- ▸(2) ['B', 'C']
```

- **arr.splice(start_index , index_count);**

  array එකකින් අවශ්‍ය index ගානක් remove කරන්න

```
> arr
<- ▸(5) ['X', 'B', 'C', 'D', 'E']
> arr.splice(1,2);
<- ▸(2) ['B', 'C']
> arr
<- ▸(3) ['X', 'D', 'E']
```

- **arr.indexOf("Value");**

  Value එකක් තියෙන index එක return කරන්න.

```
> arr
<- ▸(5) ['A', 'B', 'C', 'D', 'E']
> arr.indexOf("B");
<- 1
> arr.indexOf("z");
<- -1
```

- **arr.reverse();**

```
> arr
<- ▸(5) ['A', 'B', 'C', 'D', 'E']
> arr.reverse();
<- ▸(5) ['E', 'D', 'C', 'B', 'A']
```

- **arr.sort();**

```
> arr
<· ▸(5) ['E', 'B', 'C', 'A', 'D']
> arr.sort();
<· ▸(5) ['A', 'B', 'C', 'D', 'E']
```

Timers in JS

Single Thread Management in Browsers

Browser වල multiple threads use කරන්න access නෑ, එක thread එකයි use කරන්න පුලුවන්. Multiple processes එකම time එකක කරගන්න ඕන නම් API use කරන්න වෙනෝ.

Inbuilt functions තියෙනෝ time waiting වලට.

<u>**Timers in JS**</u>

- **setTimeout()**

  දීපු delay එක ඉවර වුනාට පස්සෙ, එක පාරක් run වෙනෝ.

```
setTimeout(function () {
    console.log("Hello there how are you.?")
}, 2000);
```

  browser එක load වෙලා 2s වලට පස්සෙ, output එක දෙනෝ.

  - clearTimeout(); ← setTimeout එක stop කරන්න පුලුවන් (Timeout එකට කලින්)

```
var testTimer = setTimeout(function () {
    console.log("Hello there how are you.?")
}, 2000);

// stop timeout
clearTimeout(testTimer);
```

- **setInterval()**

  Waiting time එකකට පස්සෙ ආයෙ ආයෙ run වෙනෝ.

```
let x = 0;
setInterval(function () {
    x++;
    console.log(x);
}, 1000);
```

- clearInterval(); ← setInterval එක stop කරනො

```
let x = 0;
var timerId = setInterval(function () {
    x++;
    console.log(x);

    if (x > 5) {
        clearInterval(timerId);// stop the set Interval
    }
}, 1000);
```

JS Data types 9

- Primitive data types 6
  - String
  - Number
  - Boolean
  - Undefined
  - Simbol
  - BigInt

- Structural data types 2
  - Objects
  - Functions

- Structural root primitive
  - null

Primitive data types 6

1. String

- “ Type1 ” - (double quotes ඇතුලෙ)
- ‘ Type2 ’ - (single quotes ඇතුලෙ)
- ` Type3 ` - (backtick ඇතුලෙ|**Template Literals|** Special with JS)
    - ${} ඇතුලෙ variable names use කරන්න පුලුවන්.

```
let x = " Hi";
console.log(`Type3 ${x}`);
```

output ⇒ Type3 Hi

String related methods

```
let sampleText = " Hello, Hi there! ";
console.log(sampleText); //  Hello, Hi there!

let a = sampleText.toUpperCase();
console.log(a); //  HELLO, HI THERE!

let b = sampleText.toLowerCase();
console.log(b); //  hello, hi there!

let c = sampleText.trim();
//දෙපැත්තෙම white spaces අයින් කරනවා
console.log(c); //Hello, Hi there!

let d = sampleText.trimRight();
//white spaces අයින් කරනවා (Right side)
console.log(d); // Hello, Hi there!

let e = sampleText.trimLeft();
//white spaces අයින් කරනවා (Left side)
console.log(e); //Hello, Hi there!

let f = sampleText.charAt(5);
//pass කරන index එකට අදාල character එක return කරනවා
console.log(f); //o
```

```
let g = sampleText.charCodeAt(5);
//pass කරන index එකට අදාල character ASCII code එක return කරනො
console.log(g); //111 (o වල ASCII code එක)

let h = sampleText.indexOf("e");
//pass කරන character එක මුලින්ම හම්බුවෙන index number එක return කරනො
console.log(h); //2

let hh = sampleText.indexOf("A");
//string එකෙ නැති character එකක් නම් -1 return කරනො
console.log(hh); //-1

let i = sampleText.substring(0,5);
//දෙන index range එකක් අතර තියෙන string කොටස return කරනො
console.log(i); // Hell

let k = sampleText.split("e");
//string array එක split කරනො array කෑලි වලට
console.log(k); // [' H', 'llo, Hi th', 'r', '! ']
```

2. Number

JS වල number type ඕින එකක් represents කරන්න තියෙන type එක number.

```
let decimal=20;
console.log(typeof decimal);//number

let binaryNumber=0b101;
console.log(typeof binaryNumber);//number

let octalNumber=0o17;
console.log(typeof octalNumber);//number

let hexaDecimal=0xA;
console.log(typeof hexaDecimal);//number

let floatingPoint=20.34;
console.log(typeof floatingPoint);//number
```

3. bigInt

ලොකුම number එකක් (9,007,199,254,740,991 ට වැඩි) දානව නම් bigInt type එක use කරනො.

```
let bigInt=100n;
console.log(typeof bigInt); //bigint
```

4. boolean

true, false

```
let isMarried = true;
```

5. Undefined

JS loose type language එකක්. Value එකක් define නොකර තියද්දි තියෙන data type එක Undefined. (type එක auto define වෙනෝ value එකක් assign වෙද්දි)

6. Symbol

වැඩිය use වෙන්නෙ නෑ.
[JavaScript Symbols]

Structural data types 2

1. Objects

named value container එකක්.

literal base object

```
let customer = {
   name:"Ushan",
   age :20,
   married: true
}
```

```
//access properties
customer.age; // dot notation
customer['age']; // bracket notation
```

* Object එකකට data dynamically add කරන්න, delete කරන්න පුලුවන්.

```
console.log(customer);
  ▶ {name: 'Ushan', age: 20, married: true}
< undefined
> customer.address = "Galle";
< 'Galle'
> console.log(customer);
  ▶ {name: 'Ushan', age: 20, married: true, address: 'Galle'}
< undefined
> delete customer.address;
< true
> console.log(customer);
  ▶ {name: 'Ushan', age: 20, married: true}
< undefined
```

* function දාන්නත් පුලුවන් dynamically.

```
customer.sampleMethod = function (){
   console.log("Hello !")
}
```

```
console.log(customer);
  ▶ {name: 'Ushan', age: 20, married: true, sampleMethod: ƒ}
undefined
customer.sampleMethod();
Hello !
```

2. Functions

වර්ග 2යි;

1) declarations

ex:

```
function greet(name) {
  console.log("Hello, " + name + "!");
}
```

```
greet("John"); // Output: Hello, John!
greet("Alice"); // Output: Hello, Alice!
```

2) expressions
   expressions are functions which are assigned to variables which return a value
   ex:

```
const add = function (a, b) {
  return a + b;
};

const result = add(3, 4);
console.log(result); // Output: 7
```

- JS වලදි Functions script එකෙ ඕන තැනක invoke කරන්න පුලුවන්.

(supports for function hoisting)

- **JS වල Method overloading වැඩ නෑ.**

```
function testMethod(id){
    console.log(`Method 1 invoked  ${id}`);
}

function testMethod(id,name){
    console.log(`Method 2 invoked ${id} , ${name}`);
}

// call methods
testMethod(10);
testMethod(10,"John");
```

output ⇒

```
Method 2 invoked 10 , undefined
Method 2 invoked 10 , John
```

(අන්තිමට තියෙන method එක call වෙන්නෙ)

- **Parameter count එක ගැන care කරන්නැ.**

ඇතුලෙ array එකක තියෙන්ො arguments කියල, property එකක් විදියට.
ඒකෙ method parameters ගැන information store කරල තියෙන්නෙ.
parameter unlimited ගානක් pass කරන්න පුලුවන්.

```
function testMethod(){
    console.log(arguments);
    //this array stores all the information about method parameters
}

testMethod();
testMethod("test");
testMethod("test",123,false);
```

output ⇒

```
▶ Arguments [callee: ƒ, Symbol(Symbol.iterator): ƒ]
▶ Arguments ['test', callee: ƒ, Symbol(Symbol.iterator): ƒ]
▶ Arguments(3) ['test', 123, false, callee: ƒ, Symbol(Symbol.iterator): ƒ]
```

Javascript Events

1. Global events

element එක ඇතුලෙන් attribute එකක් විදියට invork වෙන event එකක්.

```
<button onClick="alert('Clicked')"> Click </button>
```

Or invoke a method

```
<button onclick="sampleMethod()"> Click </button>
```

2. Specified events

document එක trough bind කරන event

```
<button id="btn">Click</button>

let e = document.getElementById('btn');
```

```
e.addEventListener('click',function (){
   alert('button clicked !')
});
```

JQuery

Javascript library එකක්
JavaScript wrap කරල හදල තියෙන්නෙ
DOM manipulation part එකෙ තියෙන complexity එක reduce කරගන්න use කරයි.

Download
https://code.jquery.com/jquery-3.7.0.min.js
& attach as a script

```
<script src="assets/js/jquery-3.7.0.min.js"></script>
```

CDN (without download)

```
<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.4/jquery.min.
js"></script>
```

- Default selector
  $();

  ```
  > $();
  <· ▸ ce.fn.init {}
  ```

  Site එකක් jquery use කරනවනම් මෙ වගෙ output එකක් එනො.

- Change the Default selector

  ```
  > let NEWSELECTER = jQuery.noConflict();
  <· undefined
  > NEWSELECTER();
  <· ▸ ce.fn.init {}
  > $();
  ⊗ ▸ Uncaught TypeError: $ is not a function
        at <anonymous>:1:1
  ```

Default selector එක ගැහුවට වැඩ නෑ. Jquery use කරනව කියල බලාගන්න බෑ. ඒකට අලුත් selecter එක ඕන.
====

JS vs jQuery

```
//select h1 and apply red color with JS
document.querySelector("h1").style.color = 'red';

//Doing the same with jQuery
$("h2").css('color','red');
```

$("h1"); ← select element

$("h1").css('color','red'); ← select element and change properties

CSS selectors use with jQuery

pseudo elements (::) හැර අනිත් ඔක්කොම selecters වැඩ.

```
$("h1");// type selector
$("#ulID")//id selector
$(".para");//class selector
$("[id]"); //attribute selector
$("*"); //Universal selector
$("div:nth-child(1)");//pseudo class
$("p::first-line");//pseudo elements // so sad not working
$("p,h1");//grouping selectors
$("section div");//descendent combinator
$("section>div");//child combinator
$("h2+p");//Adjacent combinator
$("h2~p");//General siblings combinator
```

JQuery Events

- **Formalized events**

JS වල නැවත නැවත use වෙන events ටිකක් methods විදියට හදල තියෙනෙ jquery වල.

```
click(function(){ });
```

```
dblclick(function(){ });
focus(function(){ });
blur(function(){ });
keydown(function(){ });
keyup(function(){ });
keypress(function(){ });
mousedown(function(){ });
mouseup(function(){ });
mousemove(function(){ });
```

```
$("#btn1").click(function(){
   alert("Done");
});
```

- **Non Formalized events**

on("event type", event);

```
$("#btn1").on("click",function (){
   alert("Done");
});
```

* unbind a event

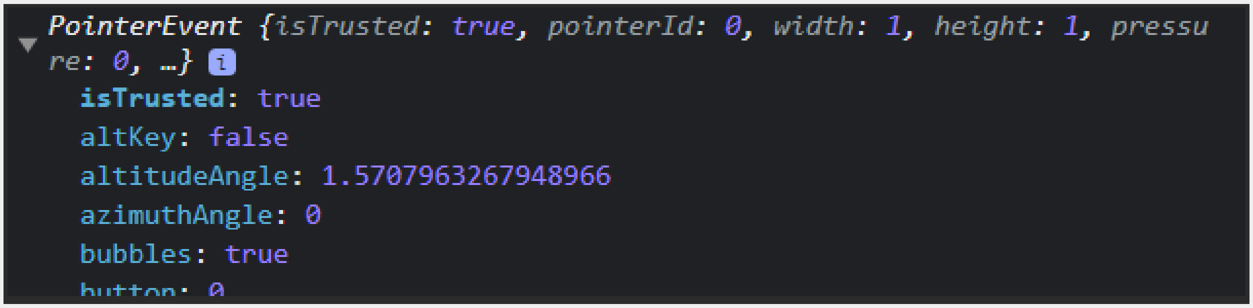
on එකෙන් event bind කරනවා වගෙ off එකෙන් event unbind කරන්න පුළුවන්, unbind කලාම event වැඩ නෑ.

```
$("#btn1").off("click");
```

* console.log(event);

```
$("#btn1").click(function(){
   console.log(event);
});
```

Output ⇒ event එක ගැන විස්තරයක් ගන්න පුළුවන්



Window & Document

🔗https://stackoverflow.com/a/9895261/19142109

window

- the main JavaScript object root, global object in a browser
- root of the document object model.
- access කරන්න පුලුවන් ⇒ `window.`
  (window. කියල access කරන්න ඕන නෑ. property name එකනේ විතරක් අඳුර ගන්න පුලුවන්)

document

- Render කරපු main object එක.(DOM - Document Object Model)
- access කරන්න පුලුවන් ⇒ `window.document`

Loaders

🔗[Loaders code]()

- **Why loaders?**

- Content එක fully load වුනාට පස්සෙ විතරක් web page එක show කරන්න.(Connection problem etc… නිසා Userට බාගෙට load වුනු දේවල් නොපෙන්නන)

- **implement a loader in JS**

  Event listeners use කරයි.

"load" and "DOMContentLoaded" Event Listener Differences

load

resources සේරම load වුනාම trigger වෙනෙො

```
window.addEventListener("load", function () {
  // Page එකයි resources ඔක්කොොම load වෙලා ඉවර වුනාම
  // resources සේරම ඕන actions perform කරන්න
});
```

DOMContentLoaded

HTML structure එක විතරක් load වුනාම trigger වෙනෙො
(සමහර images තාම load නොවී තියෙන්න පුලුවන්)

```
document.addEventListener("DOMContentLoaded", function () {
  // DOM content එක load වෙලා manipulated කරන්න ready වුනාම
  // HTML structure එක විතරක් ඕන actions perform කරන්න
});
```

* In JQuery

load

```
$(window).on("load", function() {
  // code
```

```
});
```

DOM Content Load

```
$(document).ready(function() {
  // code
});
```

Or simply,

```
$(function() {
  // code
});
```

Validations (RegEx)

(🔗 Regex ⇐ patterns)

JS වල regex use කරන විදිය

```
let regEx = /[a-z]{2}/;
```

JS වල Regular Expressions , objects විදියට සලකන්නෙ.

```
console.log(typeof regEx);
object
```

test() method එකෙන් පුලුවන් match වෙනවද නැද්ද බලන්න.
(match වෙනවනම් true, නැත්නම් false return කරනවා.)

```
regEx.test("asd");
true
regEx.test("@#$");
false
```

Traversing Methods in Jquery

DOM එකෙ තියනෙ element වලට තියනෙ relations අනුව,
අදාල elements select කරගන්න / manipulate කරන්න තියනෙ methods.

$("main").children(); ← main එකේ තියනෙ children ඔක්කොම return කරනො

$("main").children(":nth-child(1)");

$("main").children(":eq(0)"); ← returns jquery object

$("main").parent(); ← ලගම ඉන්න parent හොයාගන්න.

$("main").parents(); ← ලගම ඉන්න parentලා ඔක්කොම හොයාගන්න.

```
// jquery ekka gahanna oona jqurery witarai
// get eken return kranne dom object ekak
```