Error Handling & (finally is always Run)

Keyword's : Try(), catch(), finally, throw();

* Try, catch

આ બંને method એ છે કે Javascript માં Error Handling માટે use કરવામાં આવતી method છે.

=> જે synchronous way માં work કરે છે! એ try ની અંદર ex. asynchronous કે code આવે જેમ કે setTimeout, setInterval, Promise etc. લખવામાં આવે તો તે try method એ તેના output માટે wait કરતી નથી And a Error throw કરે છે!

=> અને આ Error Javascript દ્વારા બનાવવામાં આવતી Error છે, not we defined in catch block.

=> but :- try, catch ને asynchronous code ની અંદર લખવામાં આવે તો તે proper work કરે છે. without Error.

Valid (inside async code)

```
setTimeout(()=>{
    try {
        (start)
    }
    catch(error)
    { (error)}
}, 2000);
```

Not Valid

```
try {
    setTimeout(()=>{}, 3000)
}
catch(error) {
    (error)
}
```

=> Note:-

જ્યારે પણ try & catch લખવામાં આવે ત્યારે જો try માં Error હોય તો try first Time Run તો થાય છે ત્યાર બાદ તે catch માં Error throw કરે છે.

main types of Error

Error types in Javascript

① EvalError :-

આ Error ત્યારે show થાય જ્યારે Javascript માં eval name ના function call કરવામાં આવ્યુ હોય છે.

② RangeError :-

આ Errorમાં ત્યારે show થાય ત્યારે કે જ્યારે HTML ના input type ના Range input દ્વારા કોઈ Error genreet થતી હોય છે

* ③ Reference Error :-

Reference Error ત્યારે show થાય જ્યારે JS માં કોઈ એવા function કે Variable ને call કરવામાં આવે જે defined ના કરવામાં આવ્યો હોય છતા call થાય છે!

④ SyntaxError :-

SyntaxError માં જ્યારે કોઈ typing કે if, loop ના syntax ખામી વગર type કરવામાં ભૂલ હોય ત્યારે show થાય છે

⑤ TypeError :-

Type Error એ a Variable ના keyword જેવા કે const, let, var etc, string, boleem, number જેવા સીવાયના જો કોઈ type user દ્વારા decler કરવામાં આવ્યો હોય તો આવે છે! Error.

6 URLError :-

URl Error એટલે કે જ્યારે url bar ના pass કરવામાં આવતા Value એ wrong હોય અથવા તેની syntax wrong હોય ત્યારે ERROR આવે છે!

(7) AggregatError :-

AggregatError એ promise use કરવામાં આવે ત્યારે show થાય છે જ્યારે promise માં કોઈ Error એ જ્યારે તેને call કરવામાં આવે ત્યારે AggregatError show થાય છે.

Note

catch( ){ }; આ syntax માં પેહ Error object pass થઈ શકે છે! જેમાં ઘણા key હોય છે! જે આપણે console કરાવી શકીએ છે. (error) obj કોઈ પણ name આપીને pass થઈ શકે

1. error.name :- give's Error type name
2. error.message :- give Error txt.
3. error.stack : give full Error with name / txt and Error line.

* throw

throw method એ try માં મુકવામાં આવે છે. જે new keyword ના use Error create કરે છે. જે માં આપણે કઈ પણ આપણી રીતે Error show કરાવી શકીએ છે.

⇒ throw દ્વારા create થતી Error એ catch માં pass થાય છે

→ Example syntax :-

```
try {
    log ("Hello", x);          // x → variable
    if (!x)
    { throw new Error ('x is not defind'); }   // throw, new → keyword
}
catch(e) {                      // e → keyword, pass
    log(e); }
```

output

// x is not defind

__Proto__ or [[prototype]]

* __Proto__ એ object માટે use થતી method or trick છે જેની મદદ થી એક object નો Refrance બીજા object સાથે link કરવામાં આવે છે!

⇒ object માં Proto use કરવા માટે object ની method નો use થાય છે!

Method & Syntax

```
ObjectName = Object.Create(objectName);
```

- ObjectName → main object name
- Object.Create → method
- (objectName) → Refarence object name

⇒ Proto ની મદદથી જ્યારે mainobject પરથી કોઈ key & value call કરવામાં આવે કે પોતે ધરાવતો નથી પણ Refarence object ધરાવે છે તો તે તેની પાસે થી જ્યારે તેને inherit કરે છે!

-: Ex :-

```
obj1 = {
    key1 : "Value1",
    key2 : "Value2"
}
```

```
obj2 = {
    key3 : "Value3";
    key5 : "Value5".
}
```

```
obj2 = object.Creat(obj1);   <= tack Value from obj1

obj2 ~ "to   obj2.key3 = "Value3";
             obj2.key4 = "Value4";

Console.log(obj2.key1); // Value1
```

★ Prototype

⇒ Prototype એટલે function + object in JavaScript જ્યારે આપણે કોઈ પણ function create કરીએ છીએ ત્યારે Automatic script દ્વારા function સાથે એક prototype નામનો object create થઈ જાય છે. જે એક normal object જેવો functionality ધરાવતો હોય છે જેમાં key & Value Add કરી શકાય છે

(Prototype object use કરવા માટે ફક્ત func નુ name લખવું જરૂરી છે)

⇒ Syntax of Call prototype object

- function Student () { }; (function name)

Student.prototype.key1 = "V1"
(Add key2)

⇒ Student ના Prototype ને handel કરવા માટે dotnotation method use કરવામાં આવે છે

Syntax 2- functionName.Prototype = { } // object

Ex:-

```
function Student (name, RollN, class)
{
    Const user = object.create (Student.prototype);
    user.name = name
    user.RollNum = RollN
    user.class = class
    Return user;
}
```

(Reference proto)

⇒ Student.prototype.about() = {`Name is ${this.name}, class is ${this.class}`;}

user object પાસે હવે about() function છે જે Student / user નુ name & class print કરશે. With proto & prototype using

new keyword

→ new keyword એ basicaliy function દ્વારા ખાલી {} empty object create કરીને આપે છે. જેમાં નિજ this autometicly set કરેલ છે ~~Act~~ And એનો

Ex :-
{ 1.// new keyword એ {} empty object create કરે છે
{ 2.// object ને defalut return કરે છે.
{ 3.// object.create method એ default function ના prototype ન set કરે છે.

→
```
function Createuser (fname, lname, age)
{
        this.Firstname  = fname
        this.lastname   = lname
        this.Age        = age
}
```

→ Createuser function ને જ્યારે call કરવામાં આવશે And variable માં store કરવામાં આવશે without new keyword undifind show થશે છે!

but :-

```
Const user1 = new createuser("Hellow", "word", 21);
```
(This key & value)

```
Console.log(user1); // this re print object with
```

wit call stut

Global Execution Context :-

Closures & Memory Creation

```
1 function Myfunc(Power) {
2     return function (number) {
3         return number ** Power;
4 }
5
6 Const Square = Myfunc(2);
7 Const ans = Squrre(3);
8 Console.log(ans);
```

Global execution context ओ
आ Pashe होय छे।
1. Memory creation
2. code execution
This all are sem in function execution context to

Code execution	Memory Creation
1 to 5 line is done becouse it's function ();	window { }; this == window{}
	1. Myfunc ();
Square 6. const myfunc(2) after this squre = fu(); — CE: func() { } — MC: argue [2] power : 2 fu(); — Return with main argue	6. squre : ui / f()
7. ans = f(3) — CE: number ** Power (3²) 3 x 3 = 9 — MC: argue [3] number : 3 — return	7 ans : ui / (9)
	First part

func stop / func stop () / Global execution context call stack

Global excution Context

output
8. Console.log(ans); // 9

new keyword

→ new keyword એ basicaliy function દ્વારા ખાલી {} empty object create કરીને આપે છે. જેમાં જેનું this autometicaly set કરેલ છે At And તેનો

Ex :-
{ 1.// new keyword થી {} empty object create કરે છે
} 2.// object ને defalut return કરે છે.
{ 3.//object.create method થી defalut function ના prototype set કરે છે.

→
```
function Createuser (fname, lname, age)
{
    this.Firstname = fname
    this.lastname  = lname
    this.Age       = age
}
```

→ Createuser function ને જ્યારે call કરવામાં આવશે And variable માં store કરવામાં આવશે without new keyword undifind show થશે છે!

but :-

↓

```
Const user1 = new createuser("Hellow", "word", 21);

Console.log(user1); // this re-print object with
```

this key & value

Global Execution Context :- Closures & Memory Creation

```
line
1  function Myfunc (Power) {
2      return function (number) {
3          return number ** Power;
4      }
5
6  Const Square = Myfunc (2);
7  Const ans = Square (3);
8  Console.log (ans);
```

Global execution context ki
at Pushe ho ga
1. Memory creation
2. code execution
This all are sem in function execution context to

Code execution

Memory Creatio

window { };
this == window{}

1 to 5 line is done because it's function ();

1. Myfunc ();
6. Saure : ui | f()
7. ans : ui | (9)
first part

CE | MC
Square
6. call myfunc(2) | func () { } | argue [2]
after this | power : 2
Saure = fu(); ← Return with main argue | fu();

CE | MC
7. ans = f(3) | number ** Power (3²) | argue [3]
return 3 × 3 = 9 | number : 3

func + POP
Myfunc POP ()
Global execution context @ Call Stack

Global execution Context

output

8. Console.log (ans); // 9

DOM

Document object Model

→ Js file link in Head sesion

not we use: in Browser first run HTML file line by line then in Js file link find browser stop and run Js file.

→ Js file in body part

this is time consuming: in Browser first load HTM file then last load Js file the executed Js file

→ Js file Head with aysnc & defer atribute

* aysnc :- માં browser એ first HTML & Js file બન્નેને
<script src="" aysnc/> સાથે load કરશે જો Js file પહેલા load થશે
અને HTML file ચાલુ હશે તો browser HTML
file નુ loading stop કરી Js file ને
execut કરશે.

* defer :- browser HTML file & Js file બન્ને સાથે
<script src="" defer/> load કરશે ત્યાર બાદ જો પહેલા Js file
જ load થઈ જશે & HTML file ચાલુ હશે
તો પણ ત્યાર બાદ તે first HTML file ને
Complet load કરશે & last માં
only Js file નુ execution કરશે.

Selecct elemet

★ Select element by ID :-

document.getElementbyID ("IDName"); // only for ID

→ getElementbyID દ્વારા Tag ની ID દ્વારા Tag નું selection કરવામાં આવે છે

★ Select element using query Selector :-

document.querySelector ("#IDName, .classname, h1 (Tagname)")

→ querySelector માં CSS ની જેમ #, dot, tag name આપીને તેમને select કરવામાં આવે છે
⇒ (queryseletector ફક્ત જે first element મળે તેને જ select કરે છે બધા નહિ);

document.querySelectorAll (" same as above ");
→ querySelectorAll માં જેમ HTML માં આપેલા બધા element ને Select કરે છે (This is Return Nodelist):
⇒ Nodelist look as array but is not Array.

★ Select element using by ClassName :-

document.getElementByClassName ("className");

→ getClassName દ્વારા HTML file રહેલા element same class આપવામાં આવેલ હોય તેને select કરી શકાય.
This Return HTMLCollection it's look as Array.

★ Select with tag name :-

```
document.getElementByTagName(" Tag Name ");
```

⇒ આપણામાં આવેલા દરેક Tag ને Select Selection Return કરશે & Return માં HTMLCollection આવે છે

———— × ———— × ———— × ———— × ————

① getAttribute () Method

getAttribute method દ્વારા તેમાં આવેલ આપણામાં આવેલા tag ના Attribute ની Value Return આવે છે.

HTML :- Atribute

```
<a href="#"> Home </a>
<input type="text">
```

↑ Aribute

Js :-

```
Const link = document.getElementByTagName("a");
Console.log(link.getAttribute("href"));
```

⇒ ↳ // # ← output

```
Const input = document.getElementByTagName("input");
Console.log(input.getAttribute("type"));
```

↳ // text ← output

② Set Attribute ( ) method

→ SetAttribute દ્વારા Attribute ની Value set કરવી શકાય છે
set Attribute માં બે params લેવા છે input માટે
First = Attribute Name
Second : Attributes New Value.

Ex :- (At Name ↓, At Value ↓)

```
→ link.SetAttribute("href", "https://www.google.com/");
→ input.SetAttribute("type", "Password");
```

આના Element ની New Attribute Value set થઈ જશે.

=> Nodelist & HTML Collection in loop

① HTML Collection :-

→ HTML Collection સાથે for of loop & simpel for loop use કરી શકાય છે.

→ Becouse HTML Collection hase index value & length.

=> not use forEach methd or ~~for in loop~~

② Nodelist

→ Nodelist માં for of loop, for loop & forEach method use કરી શકાય છે.

(Nodlist માં text value હોય છે જે extra non use હોય છે ignor it)

* Converting Nodelist & HTML Collection in Array.

```
let aTag = document.getElementByTagName("a"); // Collection
let pTag = document.querySelectorAll("p"); // Nodelist

aTag = Array.from(aTag);      } pTag = Array.from(pTag);
This is now Array from        } This is now in Array.from.
```

=> Converting બાદ દરેક Array ની બધી method use કરી શકાય છે.

Page No.
Date

* getRootNode():
rootNode એ document Return કરે છે. જેમાં HTML નો <!DOCTYPE html> tag હોય છે.
જે.

```
<!DOCTYPE html>
<html>
  <head>
  </head>
  <body>
  </body>
</html>
```
} < #Document > child

* childNodes
childNode property એ Nodelist Return કરે છે.
Ex:-
let body = document.queryselector("body");

let bodydiv = body.childNode;
↑ body ના જે કોઈ tag હશે, તેની Nodelist આપશે.

* ParentNode
parentNode એ કોઈ પણ element નો parent tag Return કરે છે.
Ex:-
let body = document.queryselector("body");
let parentbody = body.parentNode;
↳ body નો parent <HTML></HTML> tag હોય છે. તે Return કરશે. (as Nodlist)

Page No.
Date

④ nextSibling

→ nexSibling આ HTML માં tag ના પછીના tag એટલે કે sibling tag ને Return કરે છે.

Ex :-

```
let head = document.querySelector("head");
let headsibling = head.nextSibling // #Text
```

{ HTML માં head tag પછી body Tag ની શરૂઆત થાય છે તો પછી nextSibling body થાવું જોઈએ.

but

nexSibling method માં nodlist ની દ્રષ્ટિએ જોઈએ તો sibling આવે છે તો Head પછી new line of space કરી તો "#text" Content આવશે body Tag નહિ.

આ problem ને Solve કરવા માટે (nextElement Sibling)

⑤ nextElementSibling

nexElementSibling એ direct Element Return કરે છે.

Ex :-

```
let head = document.querySelector("head");
let headnextElement = head.nextElementSibling; // body
```

This Returns "body";

⑥ children

chidren method આ HTMLCollection Return કરે છે જેમાં only child element જ હોય છે #Text જે content હોતા નથી.

```
let html = document.querySelector("html");
let Childes = html.chidren
```

This only Return [head & body] Tag

classList

classList method એ કોઈપણ tag & element માં આપવામાં આવેલ classList ને Array like object માં Show & Return કરે છે

↳ DOMTokenList [......];

Ex :-

→ <div class="demo box1"></div>

→
```
let div = document.querySelector(".demo");
Console.log( div.classList );
                // [ demo, box1 ];
```

(main example of use)

classList.add(), .remove(), toggel(), contains()

★ .add()

→ method એ એક class ને Add કરવા માટે use થાય છે.
element માં class Add કરે છે

Ex :-

```
div.classList.add("bg-dark");
```

in div new એક class was add "bg-dark";

★ remove():

→ આ method દ્વારા element માં કોઈ class exiest કરતો હોય And તેને remove કરવો હોય તો તેના માટે use થાય છે

Ex :-

```
div.classlist.remove("box1");
div.classlist.remove("demo");
```

Contains();

Contains method માં જો કોઈ class નુ name આપવામાં આવે and તે class selected element ને આપેલી હોય તો તે True or False value Return કરે છે

Ex:-

```
let class = div.classList.contains("box1"); // True
console.log(Class);
class = div.classList.contains("box2"); // flase
```

Toggle();

Toggle આ class જેમાં pass કરવામાં આવેલ છે તે class જો એ element ને પહેલાથી આપવામાં આવેલ હશે તો તેને remove કરશે નહીતો તેને Add કરશે.

Ex:-

```
div.classList.toggle("box1"); // remove // FirstTi
div.classList.toggle("box1"); // Add // secondTi
```

// Add HTML element & Create //

innerHTML

- innerHTML દ્વારા HTML element ની અંદર આવેલ text & તેના child element ને Target કરી શકાય છે
- innerHTML એ આખી HTML ને change કરે છે

⇒ innerHTML એ જ્યારે HTML code ને change કરવા હોય ત્યારે use કરવામાં આવે છે. HTML element ને create કરવા માટે નહિ.

Ex:-

```
let Div = document.querySelector(".box1");
Div.innerHTML = "<h2> Hello </h2>"; // Add h2 Tag
Div.innerHTML += "<h2> Hello2 </h2>";
            // This Add h2 += h2 Tag
```

* document.createElement();

આ method દ્વારા HTML માં new element create કરવા માટે use થાય છે. createElement(" "); Tag નું name લખવામાં આવે છે.

Ex :-

creating
```
const newdiv = document.createElement(" div ");
        ↓
   newdiv == <div></div>
```

#

① append() ; & appendchild();

→ element create કર્યા બાદ તેને Add કરવા માટે append method નો use થાય છે / append method એ Tag પૂરો થાય છે તેના પહેલા element ને Add કરે છે! (last માં)

② Prepend() ;

→ prepend method એ element ને Tag ના start માં element ને Add કરવા માટે use થાય છે

Ex :-

HTML	JS
start Tag ↘	let ul = document.querySelector("ul");
→ [add This]	const li = document.createElement("li");
link1	li.textContent = "new link"; // only Text
link2	// append()
[add This] ←	ul.append(li); // last of Tag
	// Prepend()
↗ End Tag	ul.prepend(li); // start of Tag

⊛ remove()

→ element ને remove કરવા માટે use થાય છે.

Ex:-

```
<ul>
  <li>...</li>
</ul>
```

```
let li1 = document.querySelector("ul li");
li1.remove();
// li is remove from ul tag
```

⊛ after & before

→ after & before એ HTML element ની બહાર બાજુ પર

→ બીજા element ને Add કરવા માટે use થાય છે.

→ before પહેલા Add કરે છે & after last માં Add કરે છે

Ex :-

befor Add

[<div> </div>

after Add

```
let div = docerment.querySelector("div")
Const h1 = docerment.createElement("h1")
// before
div.before(h1);
// after
div.after(h1);
```

} out side of div

⊛ insertAdjecentHTML("where", hTMl elem)

before = beforebegin
Prepend = afterebegin
upend = beforeend
after = afterend

} - where postion

⇒ beforebegin એ before method ને જેમ work કરે છે

⇒ afterend એ after method ની જેમ element Add કરે છે

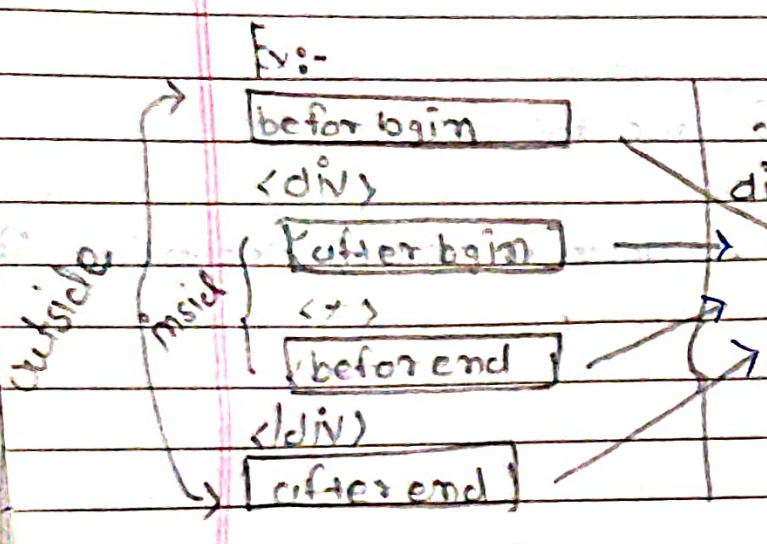
⇒ afterebegin એ prepend method ની જેમ element Add કરશે

⇒ before end = apend method Add element

Ex :- use

div

```
let div = docerment.querySelector("div");
```



clone Node :-

→ cloneNode()

cloneNode() method નો create કરેલ element નો clone કરવા માટે use થતી method છે.

Ex :-

```
→ Const ul = document.querySelector("ul");
  Const li = document.createElement("li");
  li.textContent = "new li";   ← child
  ul.apened("li");
```

→ clone

```
  Const li2 = li.cloneNode(true); // Add true fore deep clone withe child.
  ul.prepend("li2");
```

insertBefore(,);

insertBefore method માં બે prems આપવામાં આવે છે.

① માં newNode જે Add કરવાનો હોય છે તે

② જે element ની પહેલા Add કરવા માંગતા હોય તે

Ex :-

Parent ←
insertBfo
child ← text

Const ul = document.querySelector("ul");
Const li. document.querySelector("li");
Const li2 = document.createElement("li");

-: syntax :-

ul.insertBefor (li2 , li).
↑ parent ⑤ ↑ new ↑ chid old ④

④ replaceChild (newchild, oldchild);
this method use to replace child from parent.
-: syntax :-

Ex :-
Const ul = document.querySelector("ul");
Const li = document.querySelector("li");
//
Const newli = document.createElement("li");

ul.replaceChild (newli , li);
↑ new child ↑ old chid

④ removeChild ( );
→ removeChild નો પ્રયોગ કરવાથી આપણે node ને delete કરી
શકીએ.

Ex :- Const ul = document.querySelector("ul");
Const link = document.querySelector(".link");
→ ul.removeChild (link);
⇒ (ul માં જે કોઈ link" child હશે તેને તરત જ Remove
કરશે).

(A) getBoundingClientRect();

→ આ એક method છે જે આપણા element ની Height, width, top, left, xindex, y index ની info use કરી શકીએ છીએ

Ex:-

```
Const div = document.getElementByTagName('div');

Const info = div.getBoundingClientRect();
```

getBCR() થી DOMRect Return થાય છે જેમાં div ની height, x, y, width etc property મળે છે use કરવા માટે dotnotes થી use કરી શકે છે

Ex:-

```
let divheight = div.getBoundingClientRect().height;
```

↓

This is give only height

3 Way To Add Event

(I) in HTML Tag

```
<button onclick="func"> click </button>
```

function

(II) in side Js file

```
Const btn = doc.queryselector("button");
btn.onclick = function() { ... };
```

(III) with Js method.

```
Const btn = doc.querySelector('button');
btn.addEventListener("click", function);
```

Event name

function Returns

DOM Events

1. onclick - button click
2. Dblclick - button doreble click
3. mouseover -
4. mouseleave - mouse leave element
5. Keypress
6. Keydown
7. oninput
8. oninvalid

★ for Keyboard key

Keypress
Key down
} key

★ for button

onclick
Dblclick
} Target / Current Target

★ mouse

mouseover
mouseleave
} x, y

★ for input

oninput
oninvalid
} value

Events :-

⇒ mousedown / onmousedown ()

જ્યારે mouse નું button press થાય element પર ત્યારે આ Event run થાય.

⇒ mouseup / onmouseup

જ્યારે આપણે mouse button press ન થાય ત્યારે એટલે કે relesed state માં mouseup event run થાય.

⇒ mouseover & mouseout

આ બંને Event આપણે use થાય છે!

* mouseover :- જ્યારે HTML element પર mouse pointer એ HTML element પર hover or enter થાય ત્યારે જે state હોય છે તેમાં moveover Event run થાય છે
* mouse out :- જ્યારે mouse pointer એ HTML element પર થી out side પર થાય ત્યારે mouseout એ Event run થાય છે

⇒ mouseenter & mouseleave :-

આ બંને આપણે use થાય છે but not required.

* mouse enter :- when mouse pointer enter a HTML elemen This Event was run.
* mouse leave :- when mouse pointer leave HTML element This Event was run

Event

contextmenu / oncontextmenu :-

:- A user right-click on HTMLelement when This Event was run.

if we { ~~element~~ event.preventDefault() } This line stop to right-click menu open on element

* Page load event

Page load event માં Two event હોય છે

Event → ① "DOMContentLoaded" :-

જ્યારે page ની HTML download થઈને DOM Tree create થાય ત્યારે આ Event Fire થાય છે!

⇒ ② જેમાં only DOM Tree create થાય છે! External file, data, or script હજુ Download કરવાની બાકી હોય છે

Event → ② "load" :-

જ્યારે full website load થઈ જવા External element or બધા CSS files download થઈ જાય ત્યારે "load" Event Fire થાય છે

⇒ { load Event ને કોઈ HTML Element પર Add કરાવી શકાય છે }

⇒ Keyboard Event

*

① Keydown :-

આ Event આ ~~Keyboard~~ Keyboard ની keys ને press કરવામાં આવે ત્યારે run થાય છે.
આ Event ની અંદર call કરવામાં આવતા function ની અંદર window object pass કરવામાં આવે છે. જે important છે, window.key ની અંદર થી press થયેલી key આવે છે.

⇒ window object માં થી .key, .keyCode, ને use કરી શકાય છે.

② keyup :-

આ ~~digital~~ Event key press થયા બાદ જ્યારે release કરવામાં આવે ત્યારે fire થાય છે.

⇒ Scroll Event :-

① Scroll :- only scroll Event આ mouse wheel wheel Scrolling કરવા પર છે ત્યારે fired થાય છે.
Scroll આ only scroll count કરીને fired થાય છે.
when scrolling start "scroll" Event

② Scrollend :- Scrollend Event આ ત્યારે web page પર scrolling Stop થાય છે. ત્યારે જ scroll end થાય છે. અને Event Fired થાય છે.

when scrolling Stop "Scrollend" Event.

PageYoffset is deprecated
now use
[" Scrolly "]

⇒ window.PageYoffset :-

window object ની આ propertiy scroll event સાથે use થાય છે. scroll કેટલા px માં થયુ છે તે count કરવા માટે આ propertiy નો use કરવામાં આવે છે જે આપણને એક window object return કરે છે.

PageXoffset

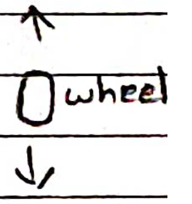
if -: use in function Syntex :-

```
window.addEventListener ("scroll", function (e) {
    if (window.pageYoffset > 100)
    {
        log(" you scroll up to 100px "); }
    else {
        log ( " you scroll les to 100px "); }
});
```

③ Wheel : આ Event પરથી જાણી શકાય કે scroll up side થયેલ છે. કે down side.

Ex :-

```
window.addEventListener ("wheel", function (e) {
    if (e.deltaY < 0) {
        log(" you scrolling up side ...."); }
    else if (e.deltaY > 0) {
        log (" you scrolling down side ..." ); }
});
```



⇒ આ Event સાથે event pass કરવામાં આવે છે. આ event objet ની properties નો use કરીને તે wheel trget કરી શકાય છે.

WheelEvent.deltaX = {
Returns double represeting the horizotal Scroll amount } ↕

wheelEvent.deltaY = {
Returns double representing the vertical scroll amount } ↔

wheelEvent.deltaZ = {
Returns dable represting the scroll amount for the z-axis } .

—: form Event :—

① focus :-
when input fild is focus this Event fired

② Blur :-
when input fild was reliesed focus this Event was Fired

③ change :-
જ્યારે input fild ની value માં કોઈ change કરવામાં આવે ત્યારે change Event fired થાય.

④ input :-
જ્યારે input fild માં કંઈ input type કરવામાં આવે ત્યારે આ input Event fired થાય.

⑤ submit :-

Submit Event એ form પર click કરવામાં આવે ત્યારે from submit કરવામાં આવે ત્યારે input ની ger કરવા માટે આ Event વપરાય છે.

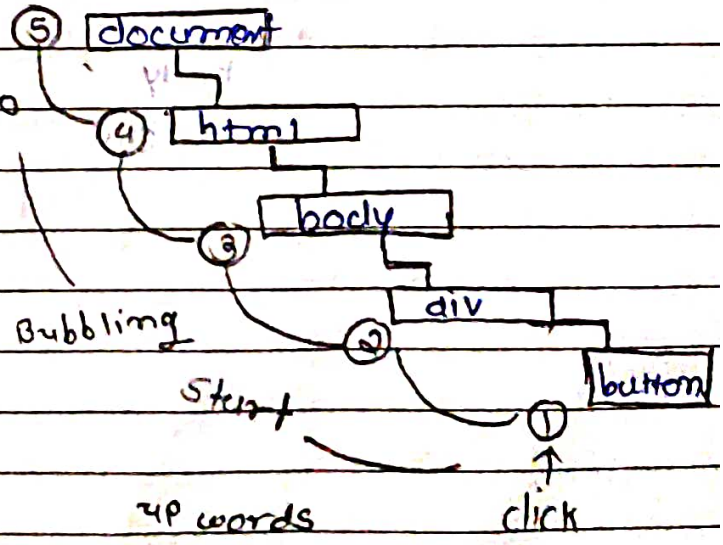
// Event bubbling && Event //

① Event bubbling :-

In the event bubbling model, an event start at the most specific element and then flows upward toward the least specific element (the document or event window);

when you click the button, the click event occurs in the following order :

① button
② div with the id container
③ body
④ html
⑤ document



Note

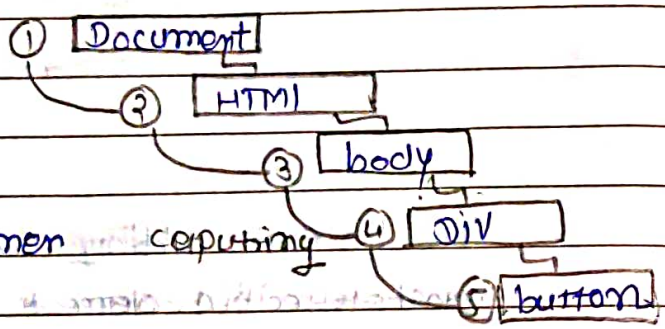
in bubbling child to parent Event Fired thai

② Event Capturing

in the event capturing modal, an event starts at the least specific element and flows downward toward the most specific element

when you click the button, the click event occurs in the following order

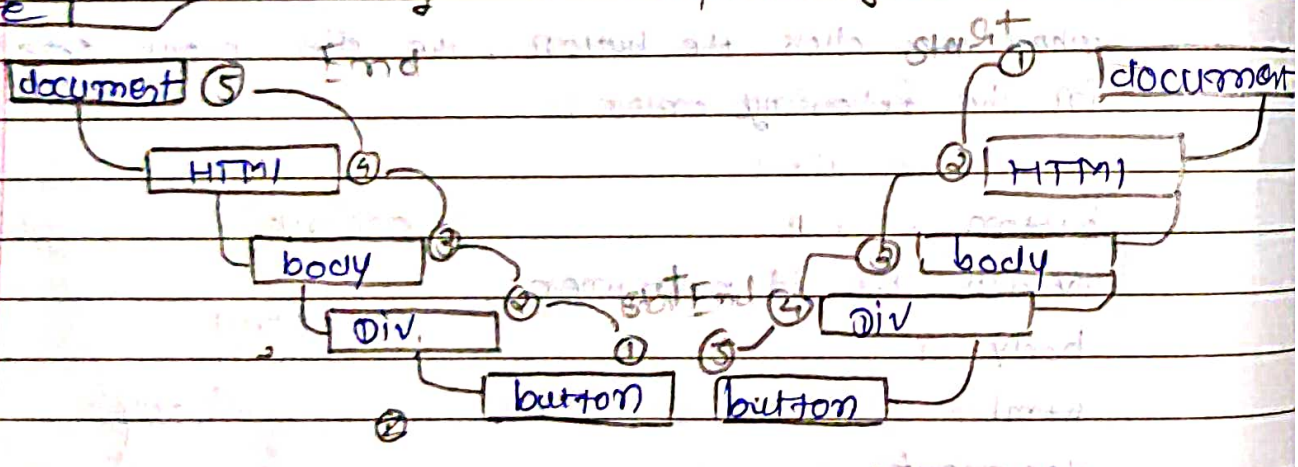
① document
② html
③ body
④ div with the id container
⑤ button



capturing
→ current to child
Event Fired via

Note

bubbling & capturing



Button
div
body
HTML
Document

Document
HTML
body
Div
button

Page No.
Date

How to add bubbling or capturing :-

addEventlister method માં three params pass કરવામાં આવે છે.

Syntax :-

addEventListener("Event", f(){}, true/fulse);

true = if true capturing ON

default ← fulse = if fulse bubbling ON

⇒ How to stop bubbing & capturing :-

આપણે bubbing & capturing ને stop કરવા કોઈ પણ function માં event object ની આ properties છે & તે function માં event object passing કરવા વાપરવામાં આવે છે.

properties is "stopPropagation();"

Ex :-

```
f(e){
  e.stopPropagation();
}
```

આ propertie only click કરવામાં આવતા Event પર call થાય છે.

Page No. | Date

function & method of eventobject

↓

PreventDefault(); -

ક્યારે પણ કોઈ browser ની default behavior chang કરવા માંગતા હોઈએ ત્યારે તે HTMLElement ને select કરી તેમાં પર def PareventDefault method use કરવામાં આવે છે।

Ex:-

stop to page Refresh
stop form submit
stop to link click etc.,

// window //

⇒ window એ object છે જે browser દ્વારા defualt create કરવામાં આવે છે! જેમાં Js ની બધી method and function save થાય છે।

⇒ window.open ( );

: Syntax :-

window.open(url , name , windowfeatures)

આમાંથી કોઈ પણ parameter pass કર્યા વગર પણ open method use કરી શકાય છે ⇒ .open();

⇒ .open() method ને કોઈ button ના click પર new window open કરવા માટે પણ use કરી શકાય છે। with click event

(I) url ⇒ window માં open કરવા માટેની url ની link add કરવામાં આવે છે

"https://www.google.com/";

② name => name એ બે button પર open() method use કરવામાં આવી હોય અને તેમના વચ્ચે toggle કરવા માટે name નો use થાય છે

=> name માં " કંઈ પણ મળી શકાય "google", "youtube" "Edge", etc

Ex: => of toggle

[button 1] [button 2]

JS

```
button1.addEl('click', function () {
    window.open( url , 'google' )
    });          (google link)    ↖— "name"
button2.addEL("click", Function () {
    window.open( url, 'google' );
  });        (Youtube link)      ↖— "name"
```

=> જ્યારે button 1 પર click થશે ત્યારે google open થશે but if we click on button2 ત્યારે જે google open હશે તો youtube open થઈ જશે. તે "name" દ્વારા Target થાય છે.

③ windowFeatures =>

આ એ new window નો Heigh, width, postion etc set કરવા માટેના Value લેવાય છે.

Ex :-

```
let url = "https://www.google.com/"
let name = 'google'
let featur = "heigh = 500, width = 500, top = 50"

window.open( url , name , featur )
```

let winID = window.open( );

=> window.open() method આ ID return કરે છે
જેના પર .close() method નો use
કરી new window ને close કરી શકાય
છે

Ex :-

```
f(){
    winID.close(); // ID
}.
```

// location //

=> Location એ window object ની એક method છે જેના થી આપણે એક website ની link રહેલ છે તેની information use and show કરાવી શકીએ છીએ

-> Location -> full object
Location.host => "127.0.0.1:5500"
.hostname => "127.0.0.1"
.Port => "5500"
... etc
url માં change પણ કરી શકાય છે.

(Deprecated) // Navigator // (read-only)

navigator એ window object ની method છે જે browser ની information provide કરે છે.

Ex :-

Navigator.appName
Navigator.appCodeName
Navigator.appVersion
} This are no use.

Page No.
Date

→ Nuvigetor. Clipboard } This method is new
Nuvigetor. bluethooth

⇒ Nuvigetor. COOKIEEnabled } This use

new topic

// closer & curriying //

Ex : of closer

:- closer ના સાથે આ old method છે જે ES6 પહેલા use થતી હતી જેની અંદર એક function ની અંદર બીજા function
⇒ Call કરવામાં આવે છે;

Ex :-

```
function Addition (A) {
  return function (B) {
    return function (C) {
      return A + B + C
    } // A+B+C(C)
  } //(B)
} //(A)
```

```
let res = Addition(1);
let data = res(2);
let Total = data(3);
=> console.log(Total);
```

6

→ :. Addition function ને call કરી ત્યારે Addtion ફંક્શન ને (A) param input તરીકે લેવ છે અને a new anoymous function return કરે છે જે (B) params argument તરીકે resive કરે છે.

:. and last માં જે function return થાય છે ને A + B + C નો Addtion કરી ને આપે છે.

* Currying new, easy to writing closer

* in closer જ્યારે આપણે કોઈ last function સુધી નુ return state કરીએ ત્યારે આપણે તેને deferent Variables માં Store કરીને પછી અંદર state arguments Pass કરતા હ
Ex:-

```
function Addition ( A ) {
    return function (B) {
        return function (C) {
            return A + B + C
        }
    }
}
```

(main function — outer function; second func — second function; Thired function — innermost function)

```
let res = Addition ( 10 );      ← This is return new second function
let res_01 = res ( 2 );         ← [ This is return Thired function ]
let res_02 = res_01 (03);       ← [ This is return value inside of Thired fun
                                    and is [ A + B + C ]
                                            10 + 2 + 3 ]
Console.log(res_02); // 15
```

*

// currying Method //

⇒ currying is new & esay way to writing close method from Es6

Ex:-

• જ્યારે closer માં Addition ની અંદર ત્રણે function માં Argu Pass કરવાના માટે એ બધા functions ને return કરી ને Variablels માં store કરી ત્યાર બાદ તેમને call કરી શકતા

Page No.
Date

જ્યારે Currying ની method માં કે Extra Variabel ને Ex :- use કરવા બદલ આ function ને use કરી શકિયે છીયે.

=> let res = Addition (10) (2) (3) ;

Here in this res Variabel is return Total of All numbers

- (10) : main func argument
- (2) : second function value or argument
- (3) : This is a third function Argument

=> console.log (res); // 15

=> =// Currying function. //=

with Arrow function

```
const Addition = (A) => (B) => (C) => {
    return A + B + C
};

let Total = Addition(10)(30)(50) ; // 90

Console.log(Total); // 90
```

Sort syntax