# DOM APIs

## Document Object Model

*javaScript*

**اسم الكاتب / عبدالرحمن عرفات المغاوري**

**Written by / Abdul Rahman Arafat ALmghawry**

**صمم هذا الكتاب ليكون مرجع لكل من تعلم ال DOM وليكون سهلاً في القراءة والفهم دون تعريب المصطلحات البرمجية لتفادي الخطأ**
**اسأل الله لكم أن ينفعكم به**

**بسم الله الرحمن الرحيم**

**( وَقُل رَّبِّ زِدْنِي عِلْمًا )**

**صدق الله العظيم**

## ما هي DOM APIs ؟

## DOM = Document Object Model

**يعني ببساطة:**
**لما المتصفح يفتح صفحة HTML بيحولها لشجرة من العناصر (Objects) اسمها DOM.**

**اما DOM APIs هي الدوال والأوامر اللي بتسمح ل JavaScript تتعامل مع عناصر الصفحة.**

**يعني تقدر بـ JavaScript:**

- **تغيير النص**
- **تغيير الألوان**
- **تضيف عناصر**
- **تمسح عناصر**
- **تتعامل مع الضغط على الأزرار**

## أهم المفاهيم والدوال

**1- Document**
**2- Node**
**3- Element**
**4- Attribute**
**5- Text Node**
**6- Parent Node**
**7- Child Node**
**8- Sibling Nodes**
**9- DOM Tree**
**10- DOM APIs**

**11- Selecting Elements**
**12- document.getElementById**
**13- document.getElementsByClassName**
**14- document.getElementsByTagName**
**15- document.querySelector**
**16- document.querySelectorAll**

**17- Traversing the DOM**
**18- parentNode**
**19- parentElement**
**20- childNodes**
**21- children**
**22- firstChild**
**23- firstElementChild**
**24- lastChild**
**25- lastElementChild**
**26- nextSibling**

**27- nextElementSibling**
**28- previousSibling**
**29- previousElementSibling**

**30- Manipulating Elements**
**31- innerHTML**
**32- textContent**
**33- style**
**34- classList**
**35- classList.add**
**36- classList.remove**
**37- classList.toggle**
**38- classList.contains**

**39- Attributes**
**40- getAttribute**
**41- setAttribute**
**42- removeAttribute**
**43- hasAttribute**

**44- Creating Elements**
**45- createElement**
**46- createTextNode**
**47- appendChild**
**48- append**
**49- prepend**
**50- before**
**51- after**
**52- insertAdjacentHTML**
**53- insertAdjacentElement**
**54- insertAdjacentText**

**55- Removing Elements**
**56- remove**
**57- removeChild**
**58- replaceChild**
**59- cloneNode**

**60- DOM Utilities**
**61- matches**
**62- closest**
**63- contains**
**64- dataset**
**65- nodeType**
**66- nodeName**

**67- Events**
**68- Event**
**69- Event Listener**
**70- addEventListener**
**71- removeEventListener**
**72- Event Bubbling**
**73- Event Capturing**
**74- Event Delegation**
**75- preventDefault**
**76- stopPropagation**
**77- DOMContentLoaded**

# 1- Document

**document ده كائن (Object) في JavaScript بيمثل كل صفحة الـ HTML.**

**ودة نقدر نقول مفتاح الوصول لباقي العناصر داخل الصفحة**

```
Document.title="my new title";

//هنا غيرنا ال title بتاع الصفحة
```

# 2- Node

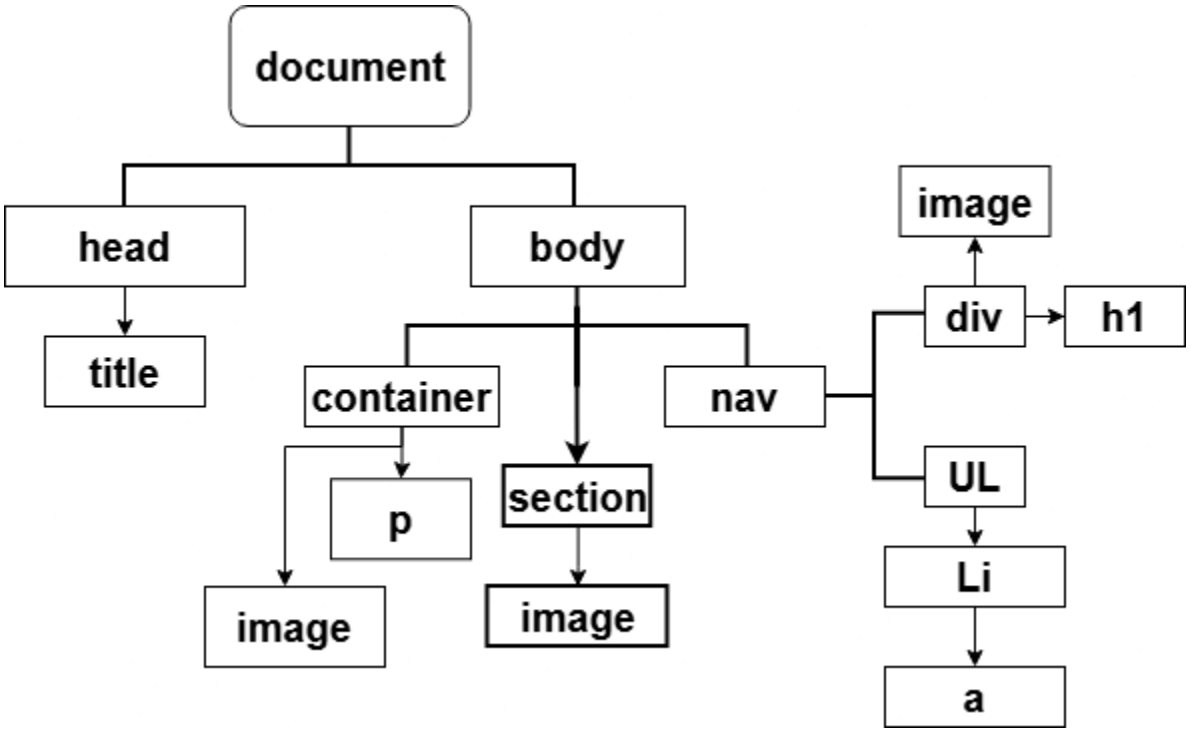
**صفحة html عبارة عن شجرة كل عنصر فيها بيسمي Node**
**هنا document بيمثل node و body بيمثل node و head بيمثل node الخ**
**الفكرة انك تعرف كل node في الـ project بتاعك جاية من اي node وتعرف ال nodes الي طالعه منها**
**ودة بيسمي dom tree**

# 3- DOM Tree

**هي شجرة بتتكون من الـ nodes , بتبدء من ال document node**

**وتنتهي بآخر node في الصفحة**

**ودة رسم توضيحي**



# 4-Element

**هو اي وسم (tag) من وسوم html موجود في الصفحة**
**(h1 , p , div ,a) وغيرها**

```
<h1>Hello</h1>
<p>Welcome</p>
```

| | |
|---|---|
| h1 | Element |
| p | Element |

# 5- Attribute

**هي خاصية تُكتب داخل الوسم الافتتاحي لعنصر HTML، وتُستخدم لإضافة معلومات أو إعدادات لهذا العنصر، مثل:**

```
<a href="#">  </a>
```

**هنا في open tag الخاص بالـ a ضيفنا**

| attribute | href |
| --- | --- |
| value | # |

# 6- Text Node

## هي أي كلمات وحروف مكتوبة بين وسمين

```
<p> i love DOM</p>
```

| | |
|---|---|
| p | element node |
| I love DOM | text node |

```
<button>click me</button>
```

| | |
|---|---|
| button | element node |
| click me | text node |

# 7- Parent Node

# 8- child Node

كلمة node قلنا إنها تمثل عنصر في الصفحة

parent node هو العنصر الذي يوجد بداخله عناصر

child node وهو العنصر الذي يوجد داخل ال parent node

```
<div>
    <p>Hello</p>
</div>
```

| | |
|---|---|
| div | Parent node |
| p | Child node |

# 9- Sibling Nodes

**لما يكون عندك مجموعة عناصر لهم نفس الأب اسمه**
**sibling nodes يعني العناصر دول اخوة**

```
<div>
    <h1>Title</h1>
    <p>Text</p>
    <button>Click</button>
</div>
```

**كلهم داخل نفس العنصر الأب بتاعهم هو div**

# 10- DOM APIs

**هي مجموعة من الدوال التي توفرها javaScript للتعامل مع صفحة html ومن خلالها نقدر نعدل عناصر الصفحة وكمان نغير خصائص css للعناصر وكمان بيها ممكن ننشئ nodes جديدة أو attributes او نعدل على الموجود ونغير خصائص وامور تانيه كتير**

**يعني باختصار**

**"هي مجموعة دوال بتستخدم للتعامل مع صفحة ال html"**

# 11- Selecting Elements

**هي الطرق التي نحدد بها عناصر من صفحة HTML عشان نعدل عليها ب**
**javaScript فيما يلي سنتناول أهمها في الشرح**

**عند تحديد العناصر من html فإنه يفضل تخزينها في متغيرات**

## 12- document.getElementById

وفي هذه الطريقة يتم استدعاء عنصر واحد من خلال ال id لأن الـ id Unique ولا يمكن تكراره فإذا كنت تريد تعديل عنصر محدد في الصفحة فيفضل اعطائة id واستدعائه من خلاله

```
let redParagraph =
document.getElementById("redParagraph");
redParagraph.style.color="red";
redParagraph.style.fontWeight="bolder";
```

عندنا صفحة html فيها براجرافات اعطينا براجراف منهم id واستدعيناه في جافا سكريبت وخزناة في متغير ومن خلال المتغير عدلنا خواص css بكل سهولة دون الحاجة لاستدعاء العنصر من جديد في كل سطر

كمان نقدر نغير ونضيف attributes أو value للعناصر

```
let go =document.getElementById("go");
go.href = "https://www.wikipedia.org/";
```

هنا عندنا link له id اسمه go استدعيناه من خلال الـ id وعدلنا ال href بتاعه

# 13 - document.getElementsByClassName

ويمكننا ايضاً استدعاء العناصر بإستخدام ال class name
فهي تمكننا من استدعاء كل العناصر الذين يملكون نفس ال class
والتعديل عليهم جميعاً ببساطة
ولكن يرجى العلم ان التعديل علي اكثر من عنصر يملكون نفس ال
class لا يكون بتلك البساطة التي تتخيلها
عند اعطاء اكثر من عنصر نفس ال class فإن JS تتعامل معهم
على أنهم items فى Array ال Array هي الـ class فإن أردت
أن تتعامل مع جميع العناصر يجب عليك ان تقوم بعمل loop على
العناصر وكذلك يمكنك تحديد عنصر محدد باستخدام ال index

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <div id="container">
            <p class="odd">1</p>
            <p class="even">2</p>
            <p class="odd">3</p>
            <p class="even">4</p>
            <p class="odd">5</p>
            <p class="even">6</p>
            <p class="odd">7</p>
            <p class="even">8</p>
            <p class="odd">9</p>
    </div>
</body>
<script src="task.js"></script>
</html>
```

```
let container = document.getElementById("container");

container.style.display="flex";
container.style.backgroundColor="gray";
container.style.width="350px";
container.style.height="350px";
container.style.flexWrap ="wrap";
container.style.alignContent ="center";
container.style.alignItems ="center";
container.style.justifyContent ="center";

let odd = document.getElementsByClassName("odd");

for (let i =0 ; i < odd.length ; i++ ) {
    odd[i].style.width = "100px";
    odd[i].style.height = "100px";
    odd[i].style.margin = "0";
    odd[i].style.backgroundColor = "black";
    odd[i].style.color = "white";
    odd[i].style.display = "flex";
    odd[i].style.fontSize = "90px";
    odd[i].style.alignItems = "center";
    odd[i].style.justifyContent = "center";
}

let even = document.getElementsByClassName("even");

for (let i =0 ; i < even.length ; i++ ) {
    even[i].style.width = "100px";
    even[i].style.height = "100px";
    even[i].style.margin = "0";
    even[i].style.backgroundColor = "white";
    even[i].style.color = "black";
    even[i].style.display = "flex";
    even[i].style.fontSize = "90px";
    even[i].style.alignItems = "center";
    even[i].style.justifyContent = "center";
}
```

# 14- querySelector()

**هي دالة تستخدم لاختيار عنصر واحد من صفحة html باستخدام محدد css**

```
let paragraph =
document.querySelector("#two");
paragraph.style.color = "red";
```

```
<p class="odd">1</p>
<p class="even" id="two">2</p>
<p class="odd">3</p>
```

## 15- document.querySelectorAll()

**هي دالة تستخدم لتحديد مجموعة عناصر تطابق محدد css معين**
**وترجع nodeList**

```
let paragraph = document.querySelectorAll(".num");
   for(i = 0 ; i < paragraph.length ; i++){
      paragraph[i].style.color = "blue";
   }
```

```
<p class="num">1</p>
<p class="num">2</p>
<p class="num">3</p>
```

## 16- document.getElementByTagName()

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

**بتاخد اي tagName زي P او a وغيرها**
**بترجع html collection زي الي بترجع من استدعاء الـ class**

# 17- Traversing the DOM

**هي عملية التنقل بين عناصر شجرة الـ DOM، حيث تُمكِّنك بعد تحديد عنصر معيّن من الوصول إلى العناصر المرتبطة به مثل العنصر الأب (Parent) أو الأبناء (Children) أو العناصر الشقيقة (Siblings) باستخدام مجموعة من الدوال في JavaScript.**

# 18- parentNode

## تستخدم للوصول لل Node الأب للعنصر الحالي

```
<div>
  <p> i love css</p>
  <p id="paragraph"> i love DOM</p>
  <p> i love javaScript</p>
</div>
```

هنا عندنا div جواة مجموعة من البراجرافات اعطينا براجراف منهم id

```
let paragraph = document.querySelector("#paragraph");
paragraph.parentNode.style.background ="red";
```

في كود js عملنا متغير و استدعينا فيه البراجراف باستخدام الـ id
بعد كدة قمنا جايبين المتغير الي فيه البراجراف واستعملنا parentNode للوصول للأب الي هو الـ div وغيرنا لونة للون الأحمر

# 19- parentElement

## تختلف أي عن parentNode ؟

| المقارنة | parentNode | parentElement |
|---|---|---|
| التعريف | خاصية تعيد العقدة الأب للعنصر | خاصية تعيد العنصر الأب فقط |
| نوع القيمة المعادة | يمكن أن تكون أي Node | تكون Element فقط |
| ما الذي يمكن أن تعيده | Element Node أو Document أو أنواع Nodes أخرى | عنصر HTML فقط |
| إذا كان الأب ليس Element | يعيده عادي | تعيد null |
| لاستخدام | عند التعامل مع كل أنواع الـ Nodes | عند التعامل مع عناصر HTML فقط |

# 20- childNodes

دي بترجعلك كل ال Nodes الي جوة العنصر على سبيل المثال كود زي دة

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
        <div id="container">
            <p> i love css</p>
            <p> i love DOM</p>
            <p> i love javaScript</p>
        </div>
    <script src="task.js"></script>
</body>
</html>
```

```
let paragraph = document.querySelector("#container").childNodes;
console.log(paragraph);
```

قد تظن للوهلة الأولى أنه هيطبع البراجرافات بس
الا انه في الحقيقه هيطلع كل الـ Nodes بالشكل دة

```
NodeList(7) [text, p, text, p, text, p, text]
  0: text
  1: p
  2: text
  3: p
  4: text
  5: p
  6: text
  length: 7
  [[Prototype]]: NodeList
```

**طب دلوقتي انا عايز اغير لون البراجرافات اخليه احمر هعمل كدة ازاي باستخدام childNodes**

**الحل:**

```
let paragraph = document.querySelector("#container").childNodes;
console.log(paragraph);

for (i = 0; i < paragraph.length; i++) {
    if(paragraph[i].nodeName === "P"){
        paragraph[i].style.color = "red";
    }
}
```

# 21- Children

Children بترجع ال Elements فقط علي عكس
childNodes الي كانت بترجع كل الـ nodes سواء
elements او غيرهم فبالتالي Children هي الأفضل عند
التعامل مع الـ Elements داخل العنصر
يلا نجرب المثال السابق ولكن باستخدام children

```
let paragraph = document.querySelector("#container").children;

console.log(paragraph);
for (i = 0; i < paragraph.length; i++) {

    paragraph[i].style.color = "red";

}
```

## 22- firstChild

دي بتجيب اول Node جوة العنصر سواء Element او Text او اي Node

## 23- lastChild

دي بتجيب اخر Node جوة العنصر سواء Element او Text او اي Node

## 24- firstElementChild

دي بتجيب اول Element جوة العنصر

## 25- lastElementChild

دي بتجيب اخر Element جوة العنصر

## 26- nextSibling

بتجيب الأخ التالي للعنصر في نفس الأب وبتجيب اي Node يعني مش Element بس ولذلك لا يفضل استعمالها عند التعامل مع Elements

## 27- previousSibling

بتجيب الأخ السابق للعنصر في نفس الأب وبتجيب اي Node يعني مش Element بس ولذلك لا يفضل استعمالها عند التعامل مع Elements

## 28- nextElementSibling

بتجيب الأخ المباشر للعنصر المشترك معاه في نفس الأب وبتجيب Element بس

```
let paragraph = document.querySelector("#paragraph").nextElementSibling;
    paragraph.style.color = "red";
```

## 29- previousElementSibling

بتجيب الأخ السابق للعنصر المشترك معاه في نفس الأب وبتجيب Element بس

```
let paragraph = document.querySelector("#paragraph").previousElementSibling;
    paragraph.style.color = "red";
```

# 30- Manipulating Elements

**التعامل مع العناصر وتعديلها**

**يعني:**

**إنت تمسك عنصر في الصفحة (زي `div` أو `p` أو `button`)**

**وبعدين تغير فيه أو تضيفله حاجة أو تمسحه أو تعدل شكله أو محتواه بالجافاسكربت**

# 31- innerHTML

## 1- بتغير الي جوة العنصر اللي احنا محددينه قبلها
## يعني بتستبدل محتوى العنصر تماماً

```
   <div id="container">
       <p id="paragraph"> i love css</p>
       <p> i love DOM</p>
       <p> i love javaScript</p>
  </div>
```

```
let paragraph = document.querySelector("#container");
    paragraph.innerHTML = "red";
```

## الناتج هيستبدل كل البراجرافات بكلمة red

## 2- كمان تقدر من خلالها تضيف elements بمعنى اكتب فيها اكواد html وهتتضاف للصفحة كعناصر html مش مجرد text

```
let paragraph = document.querySelector("#container");
paragraph.innerHTML = "<div> <p> hello world </p> </div>";
```

## دة هيضيف جوة ال div الي احنا استدعيناه div جواة براجراف جوة البراجراف hello world بالتالي اللي هيظهر في الصفحة
## hello world بس
## وليس
##   hello world  
## السطر ده هيعتبرة كود html

**3- اذا عايز تضيف عالي في العنصر دون مسح محتواه اسعمل +=**

```
let paragraph = document.querySelector("#container");
paragraph.innerHTML += "<div> <p> hello world </p> </div>";
```

**بكدة الناتج هيكون**

i love css

i love DOM

i love javaScript

hello world

## إمتى أتجنب `innerHTML`؟

**1- لو البيانات جاية من المستخدم مباشرة لأن ده ممكن يعمل مشاكل أمنية اسمها: XSS Cross-Site Scripting**

**2- لو عايز تضيف عناصر كتير بشكل احترافي**

**3- لو خايف على الأمان**

# 32- textContent

**ممكن تستعملها في :**
**تجيب النص اللي جوة العنصر**
**إذا حاولت اضافة عناصر html زي ما عملنا في innerHTML بتضيفها كنص مش هتعتبرها html**

```
<div id="box">
   <p>apple</p>
   <p>tree</p>
</div>
```

```
let box = document.getElementById("box");
console.log(box.textContent);
```

| output | apple<br>tree |
|---|---|

**تعديل النص اللي جوة العنص**

```
<div id="box"></div>
```

```
let box = document.getElementById("box");
box.textContent = "hello world";
```

| output | hello world |
|---|---|

# 33- style

**ملاحظة هامة في css بنكتب الأوامر بالشكل دة**
**background-color في جافا سكريبت بنكتب بطريقة**
**camelCase فبتكون backgroundColor**

```
document.body.style.backgroundColor = "#001D39";
ونلاحظ هنا كمان ال value مكتوبة بين علامتين تنصيص
```

**خلينا نشرح الكود دة تفصيلياً**
**هو الكائن الي بيمثل الصفحة : document**
**كل ما نتعمق أكثر داخل العناصر بنستخدمها : ( . ) dot**
**css للعنصر المختار وما بعدها خاصية css لتنسيق : style**

**css وبعدها الدوت ثم خاصية style الموضوع ببساطة ان بنكتب**
**بين علامتين تنصين value وبنعمل يساوي وبعد كدة بنعمل ال**

# 34- classList

لما يكون عندي عنصر html زي كدة

```
<div id="box" class="card active"></div>
```

فكل class بديها للعنصر بتخزن في حاجة بنسميها class list
فلو جربنا بجافا سكريبت

```
let box = document.getElementById("box");

console.log(box.classList);
```

هيطلعلك list فيها card active

# 35- classList.add

دي بتضيف class جديدة لل classList
يعني لو عندك list ليها Class وعايز تضيف ليها كمان بنستعمل
classList.add ولو عندك عنصر ملوش اي class بنستعملها

```
element.classList.add("className");
```

وممكن تضيف أكثر من class ببساطة بالطريقة دي

```
text.classList.add("red", "big");
```

# 36- classList.remove

**نفس اسلوب استخدام add بس هنا يحذف class أو أكثر على عكس add**

# 37- classList.toggle

**لو ال class موجودة في class List بتاعت العنصر يشيلها ولو مش موجودة يضيفها**
**وبتاخد class واحدة بس علي عكس add و remove**

```
box.classList.toggle("ClassName");
```

# 38- classList.contains

**بنستخدمها عشان نعرف اذا كان ضمن الـ classList بتاعة العنصر Class معين ولا لا**

```
element.classList.contains("className");
```

**بترجع true او false**

# 39- getAttribute

**بيجيب قيمة attribute معينة من العنصر**

**قولنا قبل كده ان ال attributes هي خواص بتتضاف لل open tag**
**بتاع العنصر لاضافة او تغيير خصائص العنصر**

## html

```
<a href="https://google.com" target="_blank">Google</a>
```

**الصيغة:**

```
element.getAttribute("اسم الخاصية")
```

```
element.getAttribute("href")
```

## output:

```
https://google.com
```

**لو ال Attribute مش موجودة هيرجعلك null**

# 40- setAttribute

## بتقوم بإضافة attribute أو تعديل Attribute موجودة مسبقاً

```
element.setAttribute("Attribute", "Value")
```

## تعديل Attribute موجود بالفعل

### HTML

```
<img src="old.jpg">
```

### JavaScript

```
let img = document.querySelector("img");
img.setAttribute("src", "new.jpg");
```

## إضافة Attribute جديد

### HTML

```
<img src="image.jpg">
```

### JavaScript

```
img.setAttribute("alt", "new img");
```

# 41- removeAttribute

## وظيفته بيحذف Attribute

**الصيغة:**

```
element.removeAttribute("اسم الخاصية")
```

# 42- hasAttribute

**وظيفته ايه؟**

**بيتأكد إذا كان العنصر عنده الـ attribute دي ولا لا.**

**الصيغة:**

```
element.hasAttribute("اسم الخاصية")
```

**بيرجع true او false**

# 43- createElement

الصيغة :

```
document.createElement("tagName")
```

بتعمل عنصر  جديد زي

div

a

p

```
let div = document.createElement("div");
```

كدة ال div اتعمل بس لسه مش ظاهر في الصفحة

مثال كامل لإنشاء وإضافة عنصر للصفحة:

```
let p = document.createElement("p");
p.textContent = "Hello From JavaScript";
document.body.appendChild(p);
```

# 44- createTextNode

بتضيف نص للصفحة (text node)
مش بتضيف p مش بتضيف div هي بتضيف مجرد نص

مثال:

```
let p = document.createElement("p");
let text = document.createTextNode("Hello From
JavaScript");
p.appendChild(text);
document.body.appendChild(p);
```

هنا عملنا براجراف
بعدين عملنا textNode
بعدين ضفنا ال text جوة البراجراف
بعدين ضفنا ال p لل body

# 45- appendChild

**تقوم بإضافة عنصر داخل عنصر العنصر المضاف يتم وضعه كأخر عنصر داخل العنصر الأب**

**مثال :**

```
document.body.appendChild(div);
```

# 46- append

بتستخدم عشان تضيف عنصر أو أكثر داخل عنصر
بتضيف العناصر في اخر العنصر الأب
مثال :

```
let p = document.createElement("p");
p.textContent = "Hello";

document.body.append(p);
```

## الفرق بين append و append child

### append

ممكن يضيف عنصر أو أكثر وممكن يضيف نص

### appendChild

بيضيف عنصر واحد بس ومش بيقبل نص مباشر

## مثال لإضافة أكثر من عنصر باستخدام append

```
let box = document.getElementById("box");

let h1 = document.createElement("h1");
h1.textContent = "Title";

let p = document.createElement("p");
p.textContent = "Description";

box.append(h1, p);
```

# 47- prepend

## بتضيف العناصر في بداية العنصر الأب على عكس append اللي بتضيف في الأخر

```
let container = document.querySelector("#container");
let paragraph = document.createElement("p");
let text = document.createTextNode("from append");
let paragraph2 = document.createElement("p");
let text2 = document.createTextNode("from prepend");
paragraph.append(text);
container.append(paragraph);
paragraph2.prepend(text2);
container.prepend(paragraph2);
```

**الناتج :**

from prepend

i love css

i love DOM

i love javaScript

from append

# 48- before

**بتضيف عنصر أو نص قبل العنصر الأب يعني بتضيف العنصر أو النص قبل العنصر المحدد قبلها مش جواه**

**الصيغة :**

```
element.before(item1, item2, "text");
```

**يقبل إيه؟**

**1) عنصر واحد**

```
element.before(div);
```

**2) أكثر من عنصر**

```
element.before(h1, p);
```

**3) نص عادي String**

```
element.before("Hello");
```

**4) نص + عناصر**

```
element.before("Start ", strong);
```

# 49- after

**هي نفس before بس بينهم فرق بسيط before بتضيف عناصر قبل العنصر after بتضيف بعد العنصر**

# 50- insertAdjacentHTML

دي method بنستخدمها عشان نضيف عنصر HTML للصفحة
جوة الصفحة من غير ما نمسح الـ HTML الأساسي
احياناً هي افضل من innerHTML
خليك عارف انها بتستقبل منك كود HTML كنص والمتصفح هو الي بيحولة لعنصر حقيقي

```
element.insertAdjacentHTML(position, text);
```

## text (1

بتكتب فيها HTML CODE.

## position (2

بتحدد فين هيتضاف الـ HTML بالنسبة للعنصر.

عندي Position 4 بس

## beforebegin-1

بيضيف قبل العنصر نفسه

```
let h3 = document.querySelector("h3");
h3.insertAdjacentHTML("beforebegin", "<p>befor</p>");
```

دة هيضيف البراجراف قبل ال h3

## afterbegin-2

بيضيف جوة العنصر نفسه في بداية العنصر

```
let h3 = document.querySelector("h3");
h3.insertAdjacentHTML("afterbegin", "<span> after </span>");
```

دة هيضيف span فيها كلمة after في بداية ال h3

## beforeend-3

بيضيف جوة العنصر نفسه في آخر العنصر

```
let h3 = document.querySelector("h3");
h3.insertAdjacentHTML("beforeend", "<span> after </span>");
```

دة هيضيف span في الاخر جوة ال h3 نفسه

## afterend-4

بيضيف HTML بعد العنصر

```
let h3 = document.querySelector("h3");
h3.insertAdjacentHTML("afterend", "<p> after </p>");
```

دة هيضيف paragraph بعد ال h3

# 51- insertAdjacentElement

دي Method في JavaScript
بنستخدمها عشان نضيف عنصر HTML موجود بالفعل (Element) في مكان معين بالنسبة لعنصر تاني.

يعني بدل ما نضيف كود HTML كنص
إحنا هنا بنضيف عنصر حقيقي معمول بـ `createElement()` أو عنصر موجود أصلاً.
خليك عارف انها بتستقبل منك عنصر حقيقي مش نص يعني عنصر موجود اصلاً او عنصر انت عاملة بجافا سكريبت

الصيغة العامة

```
element.insertAdjacentElement(position, elementToInsert);
```

فيها 2 باراميتر:

1) `position`

بيحدد المكان اللي هيتم فيه الإضافة

2) `elementToInsert`

العنصر اللي عايز تضيفه (لازم يكون Element)

## أماكن الإضافة (نفس بتاعة insertAdjacentHTML)

**فيه 4 أماكن:**

- `"beforebegin"` → قبل العنصر نفسه
- `"afterbegin"` → جوه العنصر في الأول
- `"beforeend"` → جوه العنصر في الآخر
- `"afterend"` → بعد العنصر نفسه

## هوريك مثال واحد انت المفروض فهمت ال POSITION في شرح insertAdjacentHTML

```
let p = document.createElement("p");
p.textContent = "Hello";

box.insertAdjacentElement("beforeend", p);
```

# 52- insertAdjacentText

بنستخدمها عشان نضيف نص عادي في مكان معين بالنسبة لعنصر

الصيغة العامة

```
element.insertAdjacentText(position, text);
```

فيها 2 باراميتر:

## position (1

مكان الإضافة

## text (2

النص اللي عايزة تضيفيه

**html**

```
<div id="box">Hello</div>
```

مثال

```
let box = document.getElementById("box");

box.insertAdjacentText("beforeend", " World");
```

الناتج

```
<div id="box">Hello World</div>
```

# 53- Removing Elements

Removing Elements هو مش دالة دة مفهوم يندرج تحته بعض الدوال زي

remove

removeChild

replaceChild

1- الدالة remove بتتحط على العنصر نفسه علشان تحذفه من الصفحة مباشرةً

الصيغة :-

```
element.remove();
```

مثال:

HTML

```
<div id="box">أنا عنصر</div>
```

javaScript

```
let box = document.getElementById("box");
box.remove();
```

هنا احنا استدعينا ال div باستخدام ال id بتاعه وبعدين حذفناه من الصفحة باستخدام remove

**2- الدالة removeChild ودي ميثود بتتحط عالأب علشان تحذف ابن منه**

**الصيغة :-**

```
parent.removeChild(child);
```

**مثال:**

## HTML

```
<ul id="list">
  <li id="item1">Item 1</li>
  <li id="item2">Item 2</li>
</ul>
```

## javaScript

```
let list = document.getElementById("list");
let item2 = document.getElementById("item2");

list.removeChild(item2);
```

**هنا احنا استدعينا ال ul من خلال الـ id**

**ثم استدعينا item2 من خلال الـ id**

**بعدين باستخدام المتغيرات الي خزنا فيها الي استدعيناه من عناصر حددنا العنصر الأب ومن خلاله حذفنا الإبن**

**3- الدالة replaceChild ودي ميثود بتتحط ع الأب علشان تستبدل عنصر ابن قديم بعنصر ابن جديد**

**الصيغة :-**

```
parent.replaceChild(newChild, oldChild);
```

**مثال:**

## HTML

```
<div id="container">
  <p id="old">نص قديم</p>
</div>
```

## javaScript

```
let container = document.getElementById("container");
let oldP = document.getElementById("old");

let newP = document.createElement("p");
newP.textContent = "نص جديد";

container.replaceChild(newP, oldP);
```

**هنا استدعينا الـ container الأب وخزناه في متغير**

**بعدين استدعينا البراجراف الأبن وخزناه في متغير**

**بعدين عملنا براجراف جديد باستخدام createElement**

**بعدين ضفنا نص للبراجراف الجديد**

**بعدين باستخدام replacechild غيرنا الإبن القديم بالجديد**

# 54-cloneNode

**هي ميثود بتعمل نسخه (clone) من العنصر موجود مسبقاً**
**النسخة مش بتظهر في الصفحة لوحدها**
**لازم بعد ما تعمل Clone تضيفها بنفسك باستخدام:**

- `appendChild()`
- `append()`
- `before()`
- `after()`

الصيغة :-

```
element.cloneNode();
```

**او**

```
element.cloneNode(true);
```

**او**

```
element.cloneNode(false);
```

## معنى true و false

### false

ينسخ العنصر فقط بدون العناصر اللي جواه

### true

ينسخ العنصر + كل الأبناء اللي جواه
وده اسمه:

## Deep Clone

مثال

### HTML

```
<div id="box">
  Hello
  <span>World</span>
</div>
```

### javaScript

```
let box = document.getElementById("box");
let copy = box.cloneNode(false);
console.log(copy);
```

### Output

```
<div id="box"></div>
```

# 55- DOM Utilities

هي أدوات/خصائص جاهزة في الـ DOM بتساعدك تتعامل مع عناصر الصفحة بسهولة
زي إنك:

- تعرف العنصر ده نوعه إيه
- تبحث عن أقرب عنصر أب
- تتأكد هل عنصر جوه عنصر تاني
- تقرأ بيانات مخصصة من الـ HTML

## 1- matches()

بتتحقق هل العنصر الحالي بيطابق Selector معين ولا لأ.

مثال:

### HTML

```
<button class="btn primary">Click</button>
```

### JS

```
let btn = document.querySelector("button");
console.log(btn.matches(".primary")); // true
console.log(btn.matches(".btn")); // true
console.log(btn.matches("div")); // false
```

## 2- closest()

بتدور على أقرب عنصر أب (أو نفسه) يطابق Selector معين.

مثال:

### HTML

```
<div class="card">
  <button class="btn">Click</button>
</div>
```

### JS

```
let btn = document.querySelector(".btn");
console.log(btn.closest(".card"));
```

النتيجة:

هيرجع عنصر الـ div اللي كلاسها card

لو مش لاقى؟

```
console.log(btn.closest(".box")); // null
```

الاستخدام:

- لما تضغط على عنصر وعايز تعرف هو موجود جوه أنهي container
- ممتاز جدًا مع الأحداث

## 3- contains()

بتتحقق هل عنصر معين بيحتوي عنصر تاني جواه ولا لأ.

مثال:

### HTML

```
<div id="parent">
  <p id="child">Hello</p>
</div>
```

### JS

```
let parent = document.getElementById("parent");
let child = document.getElementById("child");

console.log(parent.contains(child)); // true
console.log(child.contains(parent)); // false
```

الاستخدام:

- معرفة هل العنصر المضغوط جوا menu أو modal
- مفيد في إغلاق القوائم لما المستخدم يضغط براها

## dataset -4

دي خاصية بتستخدمها علشان تقرأ أو تعدل data-* attributes من الـ HTML.

HTML

```
<div id="user" data-id="101" data-role="admin"></div>
```

JS

```
let user = document.getElementById("user");

console.log(user.dataset.id); // 101
console.log(user.dataset.role); // admin
```

تعديل القيمة:

JS

```
user.dataset.id = "202";
console.log(user.dataset.id); // 202
```

**لو الاسم فيه –**

```
<div data-user-name="Ahmed"></div>
```

**JS**

```
console.log(element.dataset.userName);
```

**الاستخدام:**

- **تخزين بيانات بسيطة على العنصر**
- **ممتاز في المشاريع بدل ما تعمل متغيرات كتير**

## 5- nodeType

بتعرف نوع الـ Node ده إيه (عنصر، نص، تعليق...).

أشهر القيم:

- 1 = عنصر Element
- 3 = نص Text
- 8 = تعليق Comment
- 9 = Document

مثال:

```
let el = document.querySelector("div");
console.log(el.nodeType); // 1
```

مثال على Text Node:

```
<div>Hello</div>
let textNode = document.querySelector("div").firstChild;
console.log(textNode.nodeType); // 3
```

الاستخدام:

- لما تتعامل مع childNodes لأن فيها عناصر + نصوص + مسافات

## 6- nodeName

بترجع اسم الـ Node.

مثال:

```
let el = document.querySelector("div");
console.log(el.nodeName); // DIV
```

مع Text Node:

```
let textNode = document.querySelector("div").firstChild;
console.log(textNode.nodeName); // #text
```

مع Document:

```
console.log(document.nodeName); // #document
```

الاستخدام:

- معرفة اسم العنصر نفسه
- مفيد في الفحص أو التحقق أثناء التنقل بين العقد

---

الفرق السريع جدًا

- `matches()` → هل العنصر هو نفسه مطابق لـ selector؟
- `closest()` → هات أقرب أب مطابق

- **`contains()` → هل عنصر جواه عنصر تاني؟**
- **`dataset` → اقرأ/عدّل `data-*`**
- **`nodeType` → نوع الـ node بالأرقام**
- **`nodeName` → اسم الـ node**

# 56- Events

**ال Events هو حدث بيحصل في الصفحة**
**زي مثلاً :**
**- المستخدم ضغط على زرار**
**- حرّك الماوس**
**- كتب في input**
**- الصفحة خلصت تحميل**
**- عمل submit للفورم**

**أمثلة على الـ Events**

**`Click`**

**`mouseover`**

**`keydown`**

**`submit`**

**`change`**

**`DOMContentLoaded`**

**مثال:**

```
<button id="btn">اضغط هنا</button>
```

**لو المستخدم ضغط على الزرار ← حصل Event اسمه click**

## Event -1

**ما هو Event Object ؟**

**لما الحدث يحصل , جافا سكريبت بتبعتلك object فيه معلومات عن الحدث ده اسمه Event Object وغالباً بنسمية E أو Event**

**وبيكون فيه معلومات زي**

**- العنصر الي حصل عليه الحدث**

**- نوع الحدث**

**- مكان الماوس**

**- الزر اللي اتضغط وهكذا**

## Event LIstener -2

**يعني إيه Listener؟**

**Listener = مستمع للحدث**

**يعني كأنك بتقول للعنصر:**

**"لو حصل حدث معين، نفّذلي كود"**

**يعني:**

- **لو حصل `click` → اعمل حاجة**
- **لو حصل `input` → اعمل حاجة**
- **لو حصل `submit` → اعمل حاجة**

## addEventListiner -3

يعني اي addEventListener يعني بقول للمتصفح لما الحدث ده يحصل نفذ لي ال function دى

مثال:

### click -1

وهي انه عند الضغط على العنصر يتم تنفيذ الكود داخلها

```
const btn = document.querySelector("button");

btn.addEventListener("click", function () {
  console.log("clicked");
});
```

ليه بنستخدمها بدال onclick حسب ما تم نشره في مدونة mozilla فده لان addEventListener هي الأفضل من حيث

تقدر تضيف اكثر من listener لنفس الحدث

تقدر تتحكم في مرحلة التنفيذ (capture / bubble)

بتشتغل علي اي event target مش بس عناصر HTML او SVG

## مقارنة سريعه

### 1- onclick الطريقة القديمة

```
btn.onclick = function () {
console.log("A");
};

btn.onclick = function () {
console.log("B");
};
```

هيطبع B فقط لأن الكود التاني لغي الأول

### 2- addEventListener:

```
btn.addEventListener("click", function () {
 console.log("A");
});

btn.addEventListener("click", function () {
 console.log("B");
});
```

النتيجة:

الاتنين هيشتغلوا.

الصيغة الصحيحة لاستخدام addEventListener عندنا 3 صيغ مختلفة

```
addEventListener(type, listener)
addEventListener(type, listener, options)
addEventListener(type, listener, useCapture)
```

الموضوع ابسط مما تتخيل الي خلينا نشرح ال parameters الي تستقبلهم الدالة ببساطة واحدة واحدة

**type -1**

هو نوع ال Event الي لما يحصل الأكواد تتنفذ من أمثلتها

```
"click"
"input"
"change"
"submit"
"keydown"
"mouseenter"
"scroll"
```

مثال:

```
input.addEventListener("input", function () {
  console.log("في كتابة");
});
```

وخد بالك type بتكون case-sensitive يعني حساسة لحالة الأحرف الي هو لازم تتكتب Small Case

## Listener -2

دة اللى هيتنفذ لما الحدث يحصل

ممكن يكون:

- function
- أو object فيه handleEvent( ) method
- أو null

مثال:

```
btn.addEventListener("click", function (event) {
  console.log(event);
});
```

## 3- أو باستعمال object فيه handleEvent()

الفنكشن بتستقبل باراميتر واحد فقط غالبًا:

```
const handlerObj = {
  handleEvent(event) {
    console.log("نوع الحدث:", event.type);
  }
};

btn.addEventListener("click", handlerObj);
```

## 4- this

استعمال this في ال EventListener

```
let div = document.querySelector("#i");
div.addEventListener("click", () => {
    console.log(this);
})
```

this = العنصر اللي عليه الـ listener

الـ `event` ده فيه معلومات مهمة جدًا:

- `event.target` → العنصر اللي اتعمل عليه الحدث فعليًا
- `event.currentTarget` → العنصر اللي عليه الـ listener
- `event.type` → نوع الحدث (`click`, `input`, ...)
- `event.preventDefault()` → يمنع السلوك الافتراضي
- `event.stopPropagation()` → يوقف الانتشار
- `event.stopImmediatePropagation()` → يوقف بقية الـ listeners كمان

MDN بتأكد إن الـ callback بياخد event واحد، والـ return value بتاعه بيتُتجاهل.

## 5 - options object (مهم جدًا جدًا)

ده أقوى جزء في الصفحة.

تقدر تبعت object بالشكل ده:

```
element.addEventListener("click", handler, {
 capture: false,
 once: false,
 passive: false,
 signal: controller.signal
});
```

MDN بتعرض الأربع خيارات الأساسية دي:

.capture, once, passive, signal

خلينا نشرحهم واحدة واحد

### 1- Capture

ودي بتحدد ال Event يشتغل في ال Capture ولا ال Bubbling

يعني ايه الكلام ده؟!!

بص العنصر بيمر بعدة مراحل

1- Capture Phase (مرحلة النزول)

دي مرحلة الحدث وهو نازل من فوق لتحت لحد ما يوصل للعنصر

Document > html > body > div > button

**2- ال Target phase (الهدف)**
**لما الحدث يوصل للعنصر الي اتضغط عليه فعلاً**
**button**

**3- ال Bubble Phase (مرحلة ال Bubble)**
**بعد ما الحدث يوصل للزرار يبدأ يطلع تاني من تحت لفوق**
**Document < html < body < div < button**

**مثال :**

**HTML**

```
<div id="parent">
  <button id="child">Click</button>
</div>
```

**JS**

```
const parent = document.getElementById("parent");
const child = document.getElementById("child");
parent.addEventListener("click", () => {
  console.log("Parent");
});

child.addEventListener("click", () => {
  console.log("Child");
});
```

لو دوست علي الزرار الناتج:

```
Child
Parent
```

اذا مش ملاحظ فدة عكس ترتيب الـ CODE الي احنا كاتبينة
السبب ان الافتراضي هو Capture : False

## once -2

يعني اي ؟
يعني لو هي ترو الكود يشتغل مرة واحدة يعني لو مثلا عملت 20 click وانا مفعل once الكود اول واحدة بس اللي تحتسب
الصيغة :

```
element.addEventListener("click", handler, {
 capture: false,
 once: false,
 passive: false,
 signal: controller.signal
});
```

## passive -3

**يعني ايه؟**

**يعني بنقول اوعدك اني مش هستخدم preventDefault جوة ال listener**

**يحسن الأداء في أحداث زي:**

- `wheel`
- `touchstart`
- `touchmove`

**المتصفح يقدر يبدأ الـ scrolling أسرع بدل ما يستنى ال listener يخلص.**

**MDN بتوضح ان لو `passive: true` وبعدين حاولت تعمل `preventDefault()` → مش هيشتغل وممكن يظهر warning في الـ console. وكمان في متصفحات غير Safari الافتراضي بيكون `true` لبعض أحداث الـ wheel والـ touch لو محددتش الخيار**

## AbortController -4

**AbortController هو كائن في جافا سكريبت وظيفته يلغي عملية شغاله او يوقفها وقت ما تحب**

**اشهر استخداماته:**

**إلغاء `fetch()` (طلب API)**

**إلغاء `addEventListener()` لما تربطه بـ `signal`**

**تنظيف العمليات في React أو أي Frontend App**

**الفكرة الأساسية :**

```
const controller = new AbortController();
```

**دة بيعملك controller**

**ومن خلاله عندك**

```
controller.signal
```

**دي الإشارة (signal) اللي بتبعتها للعملية.**

**ولو عايز تلغي**

```
controller.abort();
```

## مثال مع addEventListener

```
const controller = new AbortController();

button.addEventListener("click", () => {
  console.log("clicked");
}, { signal: controller.signal });

controller.abort();
```

## Event Object -6

اول ما الحدث يحصل جافا سكريبت بتبعت **object** اسمه **Event**

```
btn.addEventListener("click", function (event) {
  console.log(event);
});
```

**ده فيه معلومات مهمة جداً زي:**

**Event.type <-** بيرجع نوع الحدث زي **CLICK**

**Event.target <-** هو العنصر الي عليه الحدث

**event.currentTarget <-** هو اب العنصر الي عليه الحدث يعني لو فيه زرار اتضغط عليه فالزرار هو **TARGET** اما ال **DIV** هو **currentTarget**

**()event.preventDefault**

**()event.stopPropagation**

## removeEventListener -7

**الصيغة:**

```
element.removeEventListener("event", functionName);
```

**1- element هو العنصر الموجود عليه ال event**

**2- event نوع الحدث**

**3- functionName نفس ال function الي اتضافت**

**مثال:**

```
const btn = document.querySelector("button");

function handleClick() {
  console.log("تم الضغط");
}

btn.addEventListener("click", handleClick);

// بعد 5 ثواني نشيل الحدث
setTimeout(() => {
  btn.removeEventListener("click", handleClick);
  console.log("تم حذف الـ event listener");
}, 5000);
```

**لازم تبعت لـ `removeEventListener()`:**

1. **نفس نوع الحدث**
2. **نفس الدالة نفسها**
3. **ولو استخدمت `capture` لازم يكون نفس القيمة**

## Event Delegation -8

**التفويض**

**بدال ما تحط Event لكل عنصر بتحط event واحدة على الأب والأب هو المفوض بمراقبة ال Event للعناصر**

**العناصر في JavaScript بتعمل Bubble يعني لو فيه زرار جوة div وضغطت الزرار الevent بيعمل bubble ويوصل للأب ومن ثم الجد وهكذا**

**يعني لو ضغطت زرار جوة ال div ال event بيوصل لل div**

### الفكرة الأساسية

### html

```
<div class="container">
  <button class="btn">1 زرار</button>
  <button class="btn">2 زرار</button>
  <button class="btn">3 زرار</button>
</div>
```

**عندنا div جواة 3 ازرار**

**ال div واخد class اسمه container**

**الأزرار واخدين class اسمه btn**

بدال ما تعمل كدة :

```
const buttons = document.querySelectorAll(".btn");

buttons.forEach((btn) => {
  btn.addEventListener("click", function () {
    console.log("Clicked");
  });
});
```

للتوضيح هنا عملنا loop علي ال class list بتاعة ال btns
علشان نحط Event علي كل عنصر جواها

تعمل كدة :

```
const parent = document.querySelector(".container");

parent.addEventListener("click", function (event) {
  if (event.target.classList.contains("btn")) {
    console.log("Clicked");
  }
});
```

هنا اضفنا addEventListener على الأب
انت ممكن تقول طب انا عايز الأزرار لية بحط ال Event علي الأب

الإجابة: علشان ال click بيعمل bubble
يعني لما تدوس علي زرار الحدث بيطلع للأب والجد لحد نهاية الشجرة

فالأب عليه addEventListener فلما الحدث بيطلع عنده بفعل ال bubble ال function بتشتغل

addEventListener هي بس بتسمع الحدث وتنفذ كود يعني بتحطها على عنصر ولما العنصر ده بيسمع الحدث بينفذ كود
لكن الأحداث هي شئ موجود فعلياً ومش بنضيفه يدوياً على كل عنصر

## 9- preventDefault

دي بتلغي التصرف الطبيعي للعنصر
يعني مثلا المفروض لو ضغطت علي لينك يفتحلك صفحة
فهي بتخليه ميفتحش الصفحة

لو عندك form وضغطت submit يبعت ال form هي بتخليه
ميبعتش ال form

وهكذا...,مثال :

```
const form = document.querySelector("form");

form.addEventListener("submit", function (event) {
  event.preventDefault();
  console.log("الفورم مش هيتبعت");
});
```

هنا ال form مش هيتبعت

## stopPropagation -10

ودي بتوقف ال bubbling بتاع الحدث يعني باختصار ال bubbling مبيطلعش فوق العنصر اللي عملناها عليه مش كان بيفضل طالع لا هو هنا بيقف عند العنصر اللي عملناها عليه

مثال

### html

```
<div class="parent">
  <button class="btn">Click Me</button>
</div>
```

### JavaScript

```
const parent = document.querySelector(".parent");
const btn = document.querySelector(".btn");

parent.addEventListener("click", function () {
  console.log("Parent clicked");
});

btn.addEventListener("click", function (event) {
  event.stopPropagation();
  console.log("Button clicked");
});
```

هنا لو ضغطت الزرار الي هيطبع Button clicked فقط ال bubble مش هتطلع للأب

## Events types -11

### 1 - Mouse Events

**click - يحدث عند الضغط ضغطة كاملة بزر الماوس الأساسي على العنصر.**

**dblclick - يحدث عند الضغط مرتين متتاليتين بزر الماوس على العنصر.**

**mousedown - يحدث عند الضغط على زر الماوس فوق العنصر.**

**mouseup - يحدث عند رفع الإصبع من زر الماوس بعد الضغط عليه.**

**mousemove - يحدث أثناء تحريك مؤشر الماوس فوق العنصر.**

**mouseenter - يحدث عند دخول مؤشر الماوس إلى العنصر بدون Bubble من العناصر الداخلية.**

**mouseleave - يحدث عند خروج مؤشر الماوس من العنصر بدون Bubble من العناصر الداخلية.**

**mouseover - يحدث عند دخول مؤشر الماوس إلى العنصر أو أحد عناصره الداخلية (يدعم Bubble).**

**mouseout - يحدث عند خروج مؤشر الماوس من العنصر أو الانتقال إلى عنصر داخلي/خارجي (يدعم Bubble).**

**contextmenu - يحدث عند فتح قائمة الزر الأيمن (Right Click Menu).**

**auxclick - يحدث عند الضغط بزر ماوس غير الأساسي مثل الزر**

الأوسط.
wheel - يحدث عند استخدام عجلة الماوس (السكرول).

## Keyboard Events - 2

keydown - يحدث عند الضغط على أي مفتاح من لوحة المفاتيح.
keyup - يحدث عند رفع الإصبع من المفتاح بعد الضغط عليه.
keypress - حدث قديم/Deprecated، كان يحدث عند ضغط مفتاح ينتج حرفًا.

## Form Events - 3

submit - يحدث عند إرسال النموذج (Form).
reset - يحدث عند إعادة تعيين النموذج لقيمه الافتراضية.
input - يحدث عند أي تغيير فوري في قيمة الحقل أثناء الكتابة أو التعديل.
change - يحدث عند تغيير قيمة الحقل ثم فقدان التركيز أو تأكيد التغيير.
focus - يحدث عند حصول العنصر على التركيز (Focus).
blur - يحدث عند فقدان العنصر للتركيز.
focusin - يشبه focus لكنه يدعم Bubble.
focusout - يشبه blur لكنه يدعم Bubble.
invalid - يحدث عند فشل التحقق من صحة حقل في النموذج.

## Window / Document Events - 4

**load - يحدث عند اكتمال تحميل الصفحة أو العنصر بالكامل.**
**DOMContentLoaded - يحدث عند اكتمال تحميل HTML بدون انتظار الصور وملفات CSS الثقيلة.**
**beforeunload - يحدث قبل مغادرة الصفحة مباشرة.**
**unload - يحدث عند مغادرة الصفحة (قديم وأقل استخدامًا).**
**resize - يحدث عند تغيير حجم نافذة المتصفح.**
**scroll - يحدث عند التمرير داخل الصفحة أو العنصر.**
**error - يحدث عند وقوع خطأ أثناء تحميل مورد أو تنفيذ سكربت.**
**hashchange - يحدث عند تغير الجزء بعد # في الرابط.**
**popstate - يحدث عند التنقل في سجل المتصفح (Back / Forward).**
**online - يحدث عند استعادة الاتصال بالإنترنت.**
**offline - يحدث عند فقدان الاتصال بالإنترنت.**

## Clipboard Events - 5

**copy - يحدث عند نسخ محتوى.**
**cut - يحدث عند قص محتوى.**
**paste - يحدث عند لصق محتوى.**

## Drag & Drop Events - 6

**drag - يحدث أثناء سحب العنصر.**
**dragstart - يحدث عند بداية سحب العنصر.**
**dragend - يحدث عند انتهاء عملية السحب.**

**dragenter - يحدث عند دخول العنصر المسحوب إلى منطقة إسقاط.**
**dragleave - يحدث عند مغادرة العنصر المسحوب لمنطقة الإسقاط.**
**dragover - يحدث أثناء مرور العنصر المسحوب فوق منطقة الإسقاط.**
**drop - يحدث عند إسقاط العنصر داخل منطقة الإسقاط.**

## Touch Events - 7

**touchstart - يحدث عند لمس الشاشة لأول مرة.**
**touchmove - يحدث عند تحريك الإصبع على الشاشة.**
**touchend - يحدث عند رفع الإصبع عن الشاشة.**
**touchcancel - يحدث عند إلغاء اللمس بسبب النظام أو سبب آخر.**

## Pointer Events - 8

**pointerdown - يحدث عند الضغط باستخدام أي أداة إدخال (ماوس / لمس / قلم).**
**pointerup - يحدث عند رفع أداة الإدخال.**
**pointermove - يحدث أثناء تحريك المؤشر أو الإصبع أو القلم.**
**pointerenter - يحدث عند دخول المؤشر إلى العنصر.**
**pointerleave - يحدث عند خروج المؤشر من العنصر.**
**pointerover - يحدث عند مرور المؤشر فوق العنصر (يدعم**

.(Bubble
pointerout - يحدث عند مغادرة المؤشر للعنصر (يدعم
.(Bubble
pointercancel - يحدث عند إلغاء التفاعل بالمؤشر.
gotpointercapture - يحدث عند التقاط العنصر للمؤشر.
lostpointercapture - يحدث عند فقدان العنصر التقاط
المؤشر.

## Media Events - 9

play - يحدث عند بدء تشغيل الوسائط.
pause - يحدث عند إيقاف الوسائط مؤقتًا.
ended - يحدث عند انتهاء تشغيل الوسائط.
volumechange - يحدث عند تغيير مستوى الصوت أو الكتم.
timeupdate - يحدث أثناء تحديث وقت التشغيل الحالي.
seeking - يحدث عند بدء الانتقال إلى وقت جديد في الوسائط.
seeked - يحدث بعد اكتمال الانتقال إلى وقت جديد.
waiting - يحدث عند توقف مؤقت بسبب التحميل.
canplay - يحدث عندما تصبح الوسائط جاهزة للتشغيل.
canplaythrough - يحدث عندما يمكن تشغيل الوسائط للنهاية
بدون توقف متوقع.
loadeddata - يحدث عند تحميل البيانات الحالية للوسائط.
loadedmetadata - يحدث عند تحميل معلومات الوسائط مثل
المدة والأبعاد.

**loadstart - يحدث عند بدء تحميل الوسائط.**
**progress - يحدث أثناء تحميل بيانات الوسائط.**
**ratechange - يحدث عند تغيير سرعة التشغيل.**
**stalled - يحدث عند توقف جلب البيانات بشكل غير متوقع.**
**suspend - يحدث عند تعليق تحميل الوسائط.**
**emptied - يحدث عند تفريغ بيانات الوسائط.**
**abort - يحدث عند إيقاف تحميل الوسائط قبل اكتماله.**

## CSS Animation Events - 10

**animationstart - يحدث عند بداية الأنيميشن.**
**animationiteration - يحدث عند تكرار دورة من الأنيميشن.**
**animationend - يحدث عند انتهاء الأنيميشن.**
**animationcancel - يحدث عند إلغاء الأنيميشن قبل اكتماله.**

## CSS Transition Events - 11

**transitionstart - يحدث عند بداية الـ Transition.**
**transitionrun - يحدث عند تجهيز الـ Transition للبدء.**
**transitionend - يحدث عند انتهاء الـ Transition.**
**transitioncancel - يحدث عند إلغاء الـ Transition قبل اكتماله.**

## Selection Events - 12

**select - يحدث عند تحديد نص داخل input أو textarea.**

**selectionchange - يحدث عند تغير التحديد في المستند.**

**13 - Fullscreen Events**
**fullscreenchange - يحدث عند الدخول أو الخروج من وضع ملء الشاشة.**
**fullscreenerror - يحدث عند فشل الدخول إلى وضع ملء الشاشة.**

**14 - Print Events**
**beforeprint - يحدث قبل بدء الطباعة.**
**afterprint - يحدث بعد انتهاء الطباعة أو إغلاق نافذة الطباعة.**

**15 - Storage Events**
**storage - يحدث عند تغيير localStorage أو sessionStorage من تبويب آخر.**

**16 - Composition Events**
**compositionstart - يحدث عند بدء إدخال نص مركب (مثل لوحات المفاتيح الآسيوية).**
**compositionupdate - يحدث أثناء تحديث النص المركب.**
**compositionend - يحدث عند انتهاء إدخال النص المركب.**

**17 - Misc Events**

**toggle - يحدث عند فتح أو غلق عناصر مثل details.**
**cancel - يحدث عند إلغاء بعض العناصر مثل dialog.**
**close - يحدث عند غلق عنصر مثل dialog.**