

WHAT IS IT?

- The DOM is a JavaScript representation of a webpage.
- It's your JS "window" into the contents of a webpage
- It's just a bunch of objects that you can interact with via JS.



```
<body>
  <h1>Hello!</h1>
  <ul>
    <li>Water Plants</li>
    <li>Get Some Sleep</li>
  </ul>
</body>
```



HTML+CSS Go In...

JS Objects Come Out



```

> document
< ▼#document
  <!DOCTYPE html>
  <html dir="ltr" lang="en" class="focus-outline-visible">
    ▶<head>...</head>
    ▶<body style="background-color: rgb(255, 255, 255);">...</body>
  </html>

> console.dir(document)

▼#document ⓘ
  URL: "chrome://new-tab-page/"
  ▶activeElement: body
  ▶adoptedStyleSheets: []
  alinkColor: ""
  ▶all: HTMLAllCollection(144) [html.focus-outline-visible, head, meta, title, style, custom-styl
  ▶anchors: HTMLCollection []
  ▶applets: HTMLCollection []
  baseURI: "chrome://new-tab-page/"
  bgColor: ""

```

SELECTING

- getElementById
- getElementsByTagName
- getElementsByClassName



Like

document

>>> gives u the html

so u have to do

console.dir(document)

>>> to get the object

Likewise

whenever you use select elements using Dom, you have to use console.dir to view the object.

It's iterable but itsn't an array so u can't use map,reduce..

querySelector

- A newer, all-in-one method to select a single element.

```
//Finds first h1 element:  
document.querySelector('h1');  
  
//Finds first element with ID of red:  
document.querySelector('#red');  
  
//Finds first element with class of  
document.querySelector('.big');
```



In query selector, we pass the “sameway of selector we pass in Css”
querySelector only selects one elements and return.

querySelectorAll

Same idea , but returns a collection of matching elements

innerHTML , textContent & InnerText

PROPERTIES & METHODS

(the important ones)

- classList
- getAttribute()
- setAttribute()
- appendChild()
- append()
- prepend()
- removeChild()
- remove()
- createElement



- innerText
- textContent
- innerHTML
- value
- parentElement
- children
- nextSibling
- previousSibling
- style

textContent: is gonna look like the how u written your html content.even though any is display:none , You can see in the textContent.

So every piece of content is gonna show.

innerHTML:what you see on the browser exactly like that. It is sensitive to what is showing on the moment .

e.g

```
const x=document.querySelector("nav a");  
x.innerText="<b> mumbai_foodbasket </b>"
```

<!-- this will not work, it will consider b tag as string.

//so use

```
x.innerHTML="<b> mumbai_foodbasket </b>"
```

innerHTML

smart

textContent

better

draw as written in .html

Not dumb, sensitive to

what is currently shown.

innerText

compact

dumb

Attributes

```
const firstlink=document.querySelector("a").src  
firstlink.setAttribute('href','https://www.udemy.com/course/the-web-  
developer-bootcamp/learn/lecture/22003782#overview')
```

```
document.querySelector('a').title  
"List of chicken breeds"  
const firstLink = document.querySelector('a')  
undefined  
firstLink.href  
"file:///wiki/List_of_chicken_breeds"  
firstLink.getAttribute('href')  
"/wiki/List_of_chicken_breeds"
```

when you use `.getAttribute()` you are getting directly from the html itself.
& when you access property directly on the element like `".href"` i.e is going through js. So we can see we got different value.

Styles

selected element object has a property i.e style

h1.style <!-- In Css we have background-color, but in the style object we dont have "-". Instead its camelcase

i.e backgroundColor

h1.style <!-- only stores inline style, not from our stylesheet

Here, you whatever you do would be inline style, you should avoid using inline style.

i.e h1.style.color=blue;

-Better way to apply styles to elements is to use a class

Issues with style property:

-inlines are bad

-using style property you can only see style attribute.

So unless you wrote the style inline, you can't read the style. I can't tell what is the `fontSize` of the element...

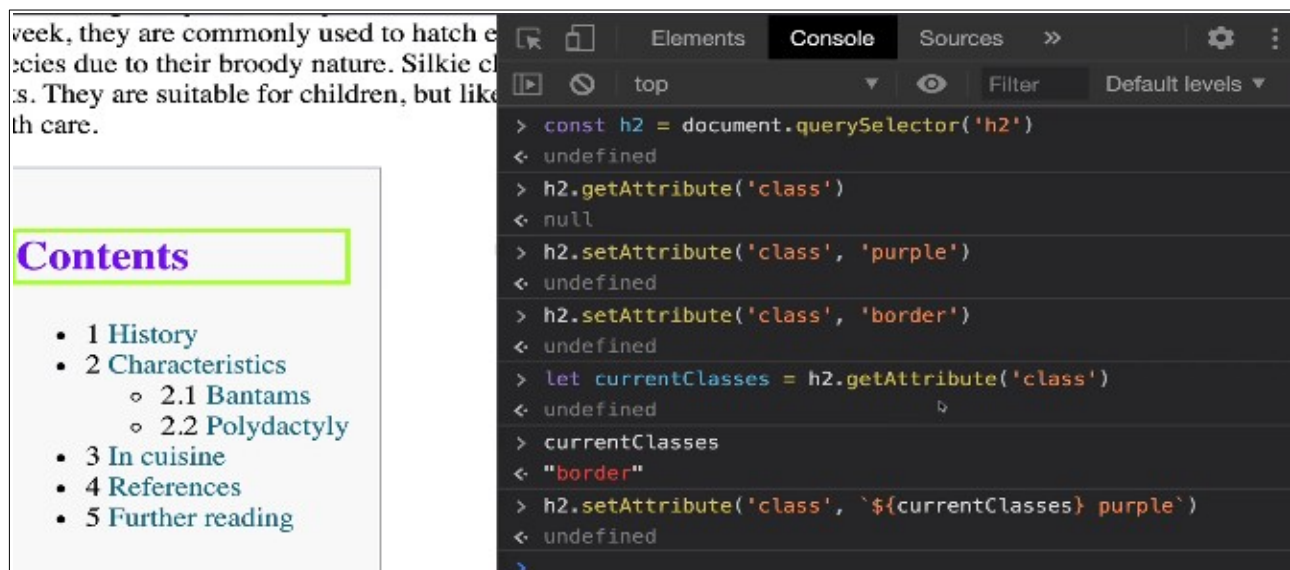
So what if I wanna make something the same font-size idk how big it is, at least by not looking at my CSS

So there is another way. To find the actual computed style once everything is applied. It's not as easy just to look at your .css copying the style over. Because you can have multiple stylesheets and they may have conflicting styles and specificity.

-You have to wait until everything is loaded and computed to figure out actual style. you can do by using

`window.getComputedStyle(h1).fontSize`

This is not a query selector, this is the actual element object in dom



The screenshot shows a web browser interface. On the left, there is a 'Contents' section with a list of links: 1 History, 2 Characteristics (with sub-links 2.1 Bantams and 2.2 Polydactyly), 3 In cuisine, 4 References, and 5 Further reading. On the right, the browser's developer console is open, showing a series of JavaScript commands and their outputs. The commands are: `const h2 = document.querySelector('h2')` (output: `undefined`), `h2.getAttribute('class')` (output: `null`), `h2.setAttribute('class', 'purple')` (output: `undefined`), `h2.setAttribute('class', 'border')` (output: `undefined`), `let currentClasses = h2.getAttribute('class')` (output: `undefined`), `currentClasses` (output: `"border"`), and `h2.setAttribute('class', `${currentClasses} purple`)` (output: `undefined`).

a week, they are commonly used to species due to their broody nature. S pets. They are suitable for children, with care.

Contents

- 1 History
- 2 Characteristics
 - 2.1 Bantams
 - 2.2 Polydactyly
- 3 In cuisine
- 4 References
- 5 Further reading

History

```
Elements Console Sources >>
top
> const h2 = document.querySelector('h2')
< undefined
> h2.classList
< ▶ DOMTokenList [value: ""]
> h2.classList
< ▶ DOMTokenList [value: ""]
> h2.classList.add('purple')
< undefined
> h2.classList.add('border')
< undefined
> h2.classList.remove('border')
< undefined
> h2.classList.contains('border')
< false
> h2.classList.contains('purple')
< true
> |
```

```
h2.classList.toggle('purple')
true
```

It might seem easier to change style using style property but by classlist it's much more simple where you can apply bunch of property by adding class or multiple classes. You can also remove class and toggle.

Traversing parent/child/sibling

i.e

parentElement

Children

nextElementSibling

previousElementSibling

Last child

LastElementChild

NextSibling

All this property allows us to navigate, traverse, move from one element to some relative or to it's parent or to it's parent parent.

```
> firstBold.parentElement
< p>...</p>
> firstBold.parentElement.parentElement
< body>...</body>
> firstBold.parentElement.parentElement.parentElement
< html lang="en">
  < head>...</head>
  < body>...</body>
</html>
> const paragraph = firstBold.parentElement
undefined
> paragraph.children
HTMLCollection(8) [b, b, a, a, a, a, a, a]
> paragraph.children[0]
< b>Silkie</b>
> paragraph.children[0].parentElement
< p>...</p>
```

nextSibling and previousSibling is gonna give us a node.

For e.g inside <h1>...</h1> we have a text node.

Some browser make new lines a text node.

| | |
|---|--|
| <p>is unknown exactly where or when these fowl with their singular
ministration of attributes first appeared, but the most well documented
ent of origin is ancient China (later another occasionally encountered
one for the bird, Chinese silk chicken), other places in Southeast Asia
have been named as possibilities, such as India and Java.^[2] The earliest
relying written account of Silkies comes from Marco Polo, who wrote
of a "furry chicken" in the 13th century during his travels in Asia.^[3] In
1986, Ulisse Alenwardi, a writer and naturalist at the University of
Alaska, Italy, published a comprehensive treatise on chickens which is
still read and admired today. In it, he spoke of "wool-bearing chickens"
of ones "clothed with hair like that of a black cat".^[4]</p>  | <pre>Silky bantam.jpg" alt> > squareImg.nextSibling < #text > squareImg.nextSibling < #text > squareImg.previousSibling < #text > squareImg.nextElementSibling < > squareImg.previousElementSibling < < p>...</p></pre> |
|---|--|

createElement

```
const newImg=document.createElement('img')
console.dir('newImg')    <!--to view
newImg.src="<someurl>"
document.body.appendChild(newImg)
```

```
const newH3 = document.createElement('h3')
undefined
newH3
<h3></h3>
newH3.innerText = 'I am new!'
"I am new!"
document.body.appendChild(newH3)
<h3>I am new!</h3>
```

appendChild

```
> const p = document.querySelector('p')
< undefined
> p.append('i am new text yaaaaaayyy!!!')
< undefined
> p.appendChild('i,am new text yaaaaaayyy!!!')
✖ ▶ Uncaught TypeError: Failed to execute 'appendChild' on 'Node': parameter 1 is not of type 'Node'.
    at <anonymous>:1:3
>
```

```
p.append('i am new text yaaaaaayyy!!!',
'asdasdasdasd')
```

It allow us to insert more than one thing at a time

```
const newB = document.createElement('b')
undefined
newB.append('Hi!')
undefined
newB
<b>Hi!</b>
p.prepend(newB)
undefined
```

y.append(div1,div2)

Element.insertAdjacentElement [click](#)

Parameters

position

A `DOMString` representing the position relative to the `element`; must be one of the following strings:

- `'beforebegin'`: Before the `element` itself.
- `'afterbegin'`: Just inside the `element`, before its first child.
- `'beforeend'`: Just inside the `element`, after its last child.
- `'afterend'`: After the `element` itself.

```
> const h2 = document.createElement('h2')
< undefined
> h2.append("Are adorable chickens")
< undefined
> h2
< <h2>Are adorable chickens</h2>
> const h1 = document.querySelector('h1')
< undefined
> h1.insertAdjacentElement('afterend', h2)
< <h2>Are adorable chickens</h2>
> h1.nextElementSibling
< <h2>Are,adorable chickens</h2>
```

```
· const h3 = document.createElement('h3')
· undefined
· h3.innerText = 'I am h3';
· "I am h3"
· h1.after(h3)
· undefined
```

```

const firstLi = document.querySelector('li')
undefined
firstLi
▶<li class="toclevel-1 tocsection-1">...</li>
const ul = firstLi.parentElement
undefined
ul
▶<ul>...</ul>
ul.removeChild(firstLi)
▶<li class="toclevel-1 tocsection-1">...</li>
|

```

Another way

```
firstLi.parentElement.removeChild(firstLi)
```

<!-- This works but isn't considered best practice

Another way

```

const img = document.querySelector('img')
undefined
img.remove()
undefined

```

EVENTS

Responding to
user inputs
and actions!



A SMALL TASTE

- clicks
- drags
- drops
- hovers
- scrolls
- form submission
- key presses
- focus/blur



- mouse wheel
- double click
- copying
- pasting
- audio start
- screen resize
- printing

Inline event

for e.g

```
<button onclick="alert('you clicked a button')">click Me </button>
```

If you wanna write multiple lines in you have to make seperation using “;”
i.e

for double click

just like in linux cmd

```
<button ondblclick="alert('you clicked a button') ; alert('stop clicking me')">click  
Me </button>
```

```

btn.onclick = function () {
  console.log("YOU CLICKED ME!")
  console.log("I HOPE IT WORKED!!")
}

function scream() {
  console.log("AAAAHHHHH");
  console.log("STOP TOUCHING ME!")
}

btn.onmouseenter = scream;

```

See Above, u r passing the function on the event but you aren't executing.

addEventListener

Specify the event type and a callback to run

```



const button = document.querySelector('h1');

button.addEventListener('click', () => {
  alert("You clicked me!!")
})

```

Not all events works on every element . For e.g keypress doesn't work on a button.

You wanna use addEventListener over onclick/..... cuz you can't have two callback functions on the same event for e.g click.

| | |
|---|---|
| mybtn.onclick=sayMyName; | Doesnt work |
| mybtn.onclick=sayMyNickName; |  only listen for last event |
| ----- | |
| mybtn.addEventListener('click',sayMyName) |  Here, both events are listening |
| mybtn.addEventListener('click',sayMyNickName) | |

```

tasButton.addEventListener('click', twist, { once: true })
tasButton.addEventListener('click', shout)

```

Tip: when you set property

document.body.backgroundColor='rgb(200,255,300)'; <!-- key:value(str)

```
const buttons = document.querySelectorAll('button');

for (let button of buttons) {
  button.addEventListener('click', function () {
    button.style.backgroundColor = makeRandColor();
  });
}
```

```
for (let button of buttons) {
  button.addEventListener('click', colorize)
}

const h1s = document.querySelectorAll('h1');
for (let h1 of h1s) {
  h1.addEventListener('click', colorize)
}

function colorize() {
  this.style.backgroundColor = makeRandColor();
  this.style.color = makeRandColor();
}
```

The keyword 'this' when You have it inside a callback ,that is invoked by some event handler, 'this' refers to the element the eventlistener was listening.

If u change ur language settings, so where your is w,a,s,d maybe anywhere. But if i use 'code' it doesn't matter. So if we care about position on keyboard use 'code'. But what you care about is the 'char', use 'key'. Again, if you dont care which button you pressed to generate the char use 'key' then.

Keyloggers

Use window.Eventlistener()

```
window.addEventListener('keydown', function (e) {
  switch (e.code) {
    case 'ArrowUp':
      console.log("UP!");
      break;
    case 'ArrowDown':
      console.log("DOWN!");
      break;
    case 'ArrowLeft':
      console.log("LEFT!");
      break;
    case 'ArrowRight':
      console.log("RIGHT!");
      break;
    default:
      console.log("IGNORED!")
  }
})
```



```
document.querySelector('input').addEventListener('keydown', function (e) {  
  console.log("KEYDOWN")  
})
```

KeyboardEvent {isTrusted: true, key: "q", code: "KeyQ", location: 0, ctrlKey: false, ...}

keyCode: 81
code: "KeyQ"

key: "q"

| | |
|-------|------|
| a | Key |
| KeyA | Code |
| | Key |
| Space | code |

Shift

Key

ShiftLeft

Code

```
<form action="/dogs">
  <input type="text" name="username" placeholder="username">
  <input type="text" name="tweet" placeholder="tweet">
  <button>Post</button>
</form>
```

```
const tweetForm = document.querySelector('#tweetForm')
tweetForm.addEventListener('submit', function (e) {
  console.log("SUBMIT!!")
  e.preventDefault();
});
```

if action isn't specified it will send data to the page you are on right now.

In below, put `e.preventDefault()` after begin inside eventlistener.

```
const tweetForm = document.querySelector('#tweetForm');
tweetForm.addEventListener('submit', function (e) {
  // const usernameInput = document.querySelectorAll('input')[0];
  // const tweetInput = document.querySelectorAll('input')[1];
  const username = tweetForm.elements.username.value;
  const tweet = tweetForm.elements.tweet.value;

  // console.log("SUBMIT!!")
  e.preventDefault();
});
```

Tip: The form is submitted by pressing 'Enter' not just by clicking the Submit button.

Input & Change events

There is copy/paste using mouse or voice over utility, so there are different ways of updating a input and keydown/keyup doesn't encompass all.

EVENT

change:-only fires when you go in and out of input element **that is** only if the input was different before you enter the input.

input:-everytime there is a change not just when you go in or out of an input. For e.g: shift or arrow keys wont fire this event. If you cut and paste that will fire this event.

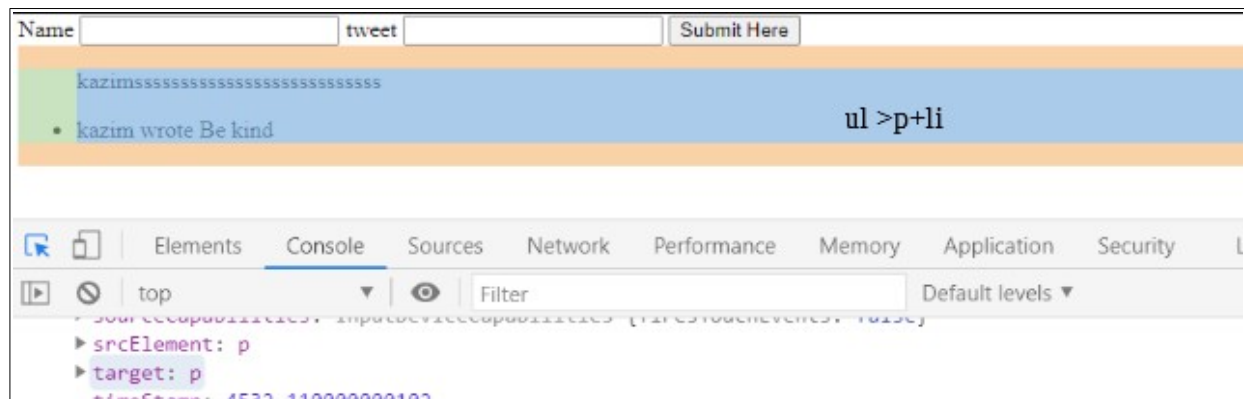
Event bubbling

```
<body>
  <section onclick="alert('section clicked')">
    <p onclick="alert('paragraph clicked')">
      I am a paragraph:
      ..... <button onclick="alert('button clicked')">Click</button>
    </p>
  </section>
```

Above, button triggered > para triggered > section triggered.

-Use `e.stopPropagation()`;

Event Delegation



Above, on li we have listener which on click are removed. But above if we submit and create a new li, it will not have a listener. So here, we placing a listener on ul.

It means we gonna add event listener to some element that is a parent. Here, We gonna listen click on ul but specifically on li that is in ul. So even though i have eventListener on ul, but the target is p here.

```
myul.addEventListener("click",(e)=>{
  e.target.nodeName==='LI' && e.target.remove();
});
```

Working with select

```
myselect.addEventListener("change",function(){
  // console.dir(e.target.options.selectedIndex);
  // max=e.target[e.target.options.selectedIndex].value;
  max=this.value;
  console.dir(this);
});
```