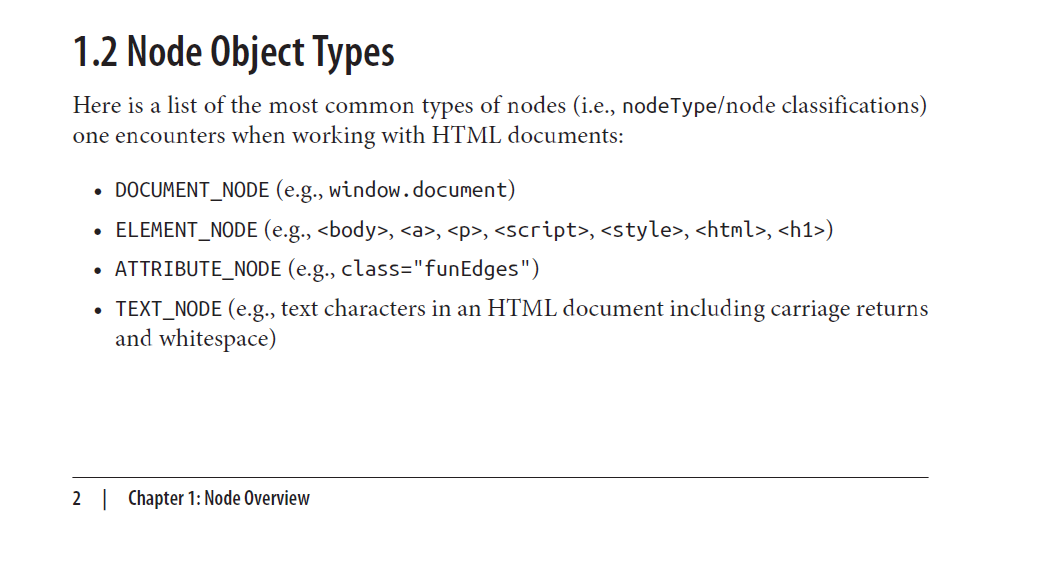
**What are Nodes ?**

When we write any html, the browsers render that and make a DOM tree consisting of **Nodes**



There are different types of Nodes

First, the topmost document itself is a node

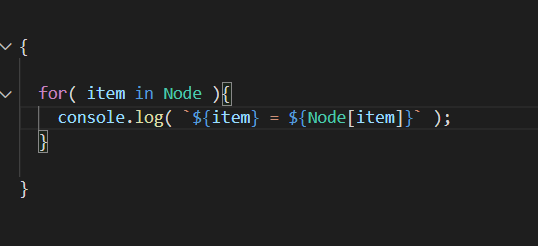
Every elements are also nodes

Attributes are also nodes

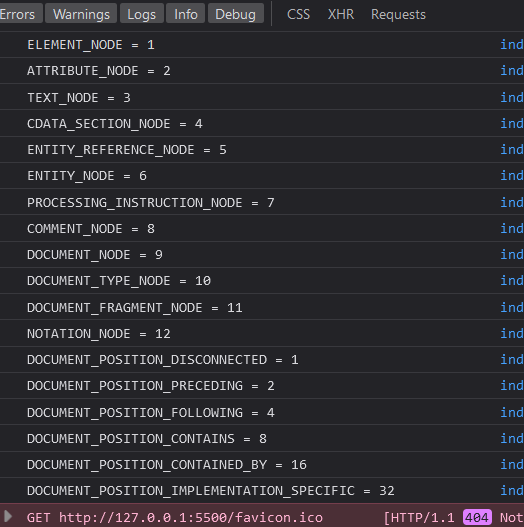
Text characters, text, whitespace are also nodes

Different Node types are represented by numbers we can actually see them

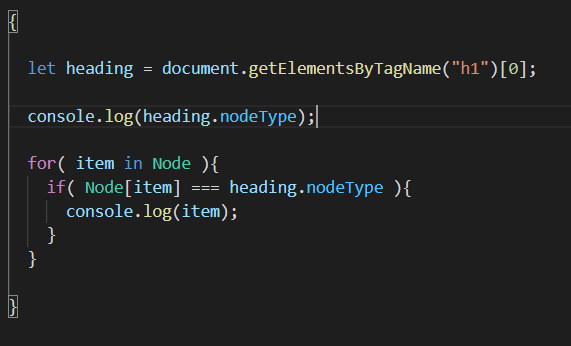
like

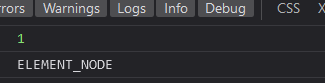


“Node” is a built in object in JS all other nodes inherit certain methods and properties from it



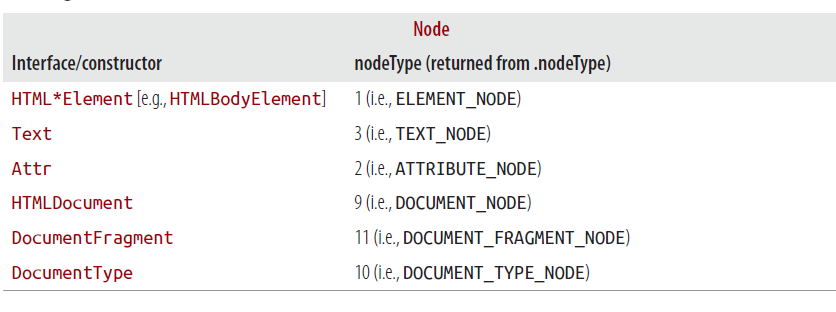
We can see the Node type of any node we want





Not all of these node types are used

These are the most frequently used ones



By just looking at the HTML, visualize the DOM tree and all the nodes

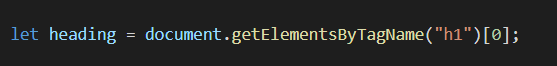
like

from the lens of HTML it is just an <h1> element or we can also say it’s a <h1> tag



But from the lens of javascript and DOM

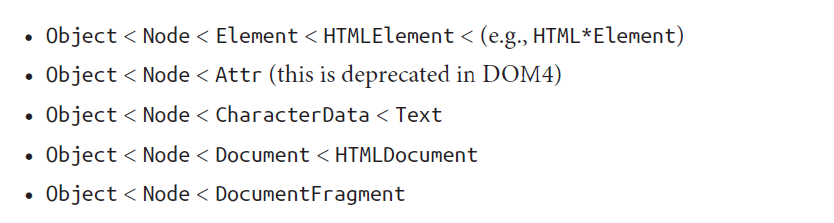
We will do this



And now heading is our node

node object to be more precisely

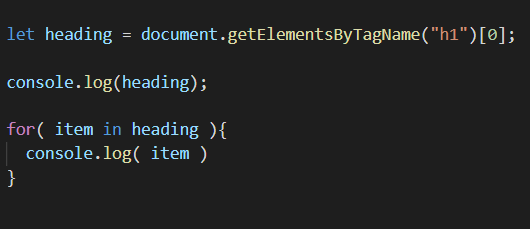
Actually there is inbuilt  object in Javascript and all other nodes inherit various properties from object

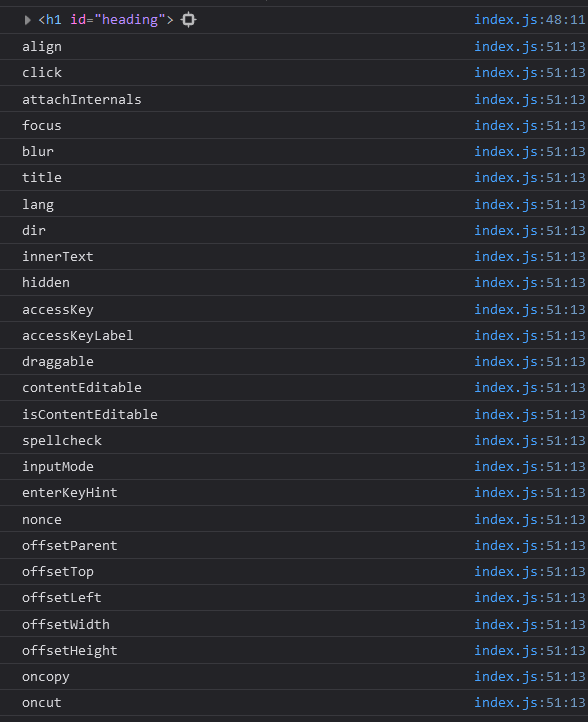


HTML Element node, Attribute node, Character Data node… … every node inherits something from “Node” object

Also we can see “Node” object also inherit something from “Object” object of javascript

**As we have been discussing, all node objects (e.g., Element, Attr, Text, and so on) inherit properties and methods from a primary Node object.**

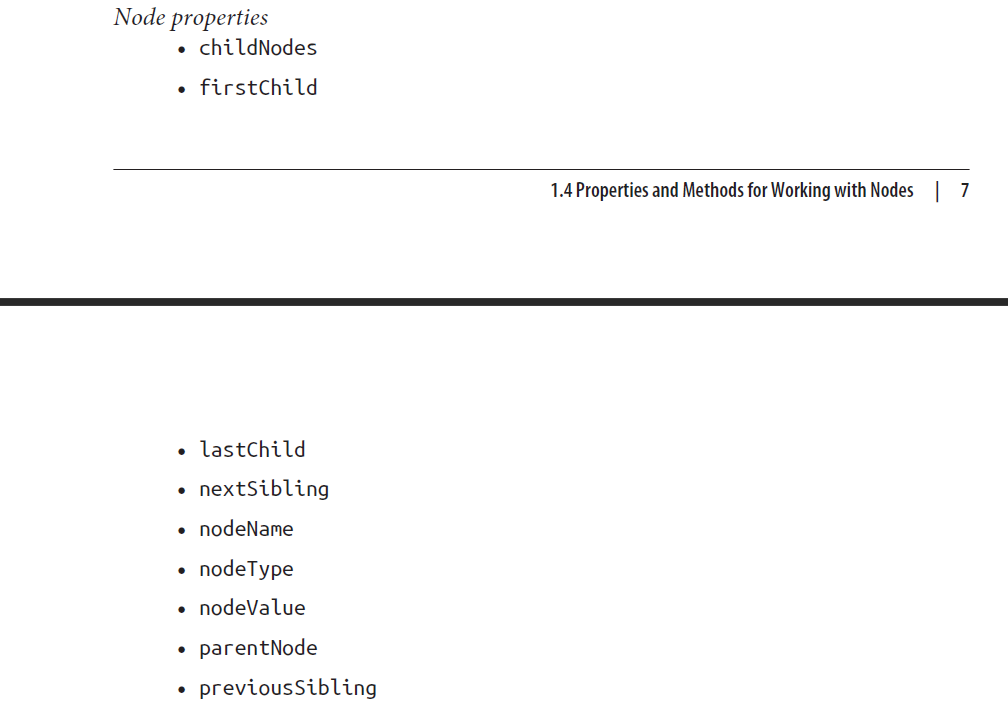




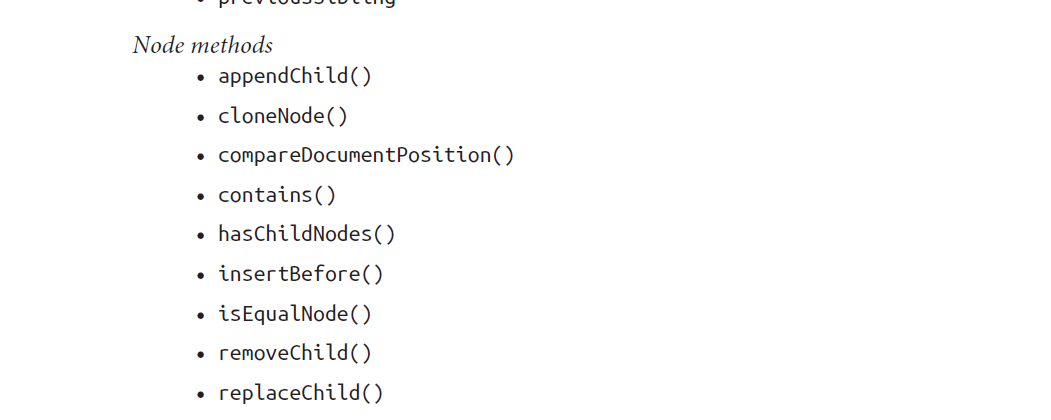
See “heading” node alone has this many properties all because of that long inheritance chain, actually as per above code all these are keys

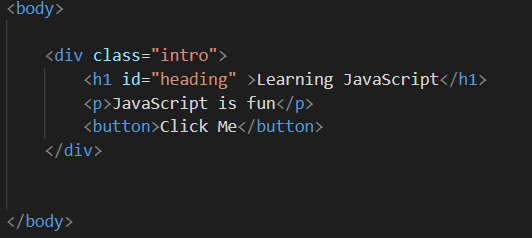
**We have provided various properties and methods to work with the nodes**

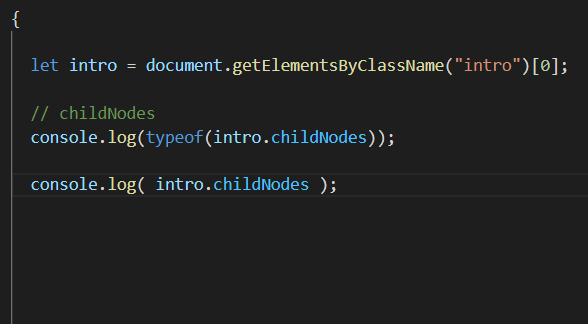
Properties like



And Methods like









We see that

.childNodes property returns us the object, having numbers starting from 0 as keys

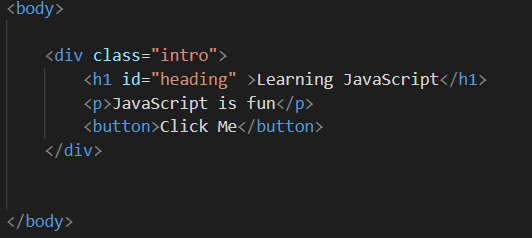
The advantage of using numbers as keys is that we can access the value like we do in arrays

like this



When we use numbers as keys in object, using “” is optional, but for other than numbers as keys using “” is compulsory

From now on whenever see HTML document like this imagine the respective DOM tree with the respective nodes



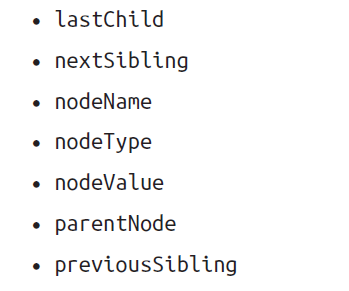


Like here

* First there is a body at top
* It has three child nodes, first one is test node \n i.e new line, second one is div and last one is also a text node i.e new line
* Actually body may have more than three childNodes, depending upton the number of \n. What I wanted to tell was, just we see a <div> inside <body> that doesnot mean <body> has only one as childNode
* Now we can actually see <div> has 7 child nodes
* See in above console output

Actually looking at the DOM tree we can easily visualize the properties like

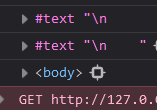




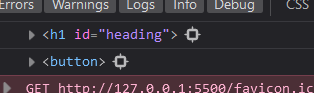
See this and visualize DOM tree

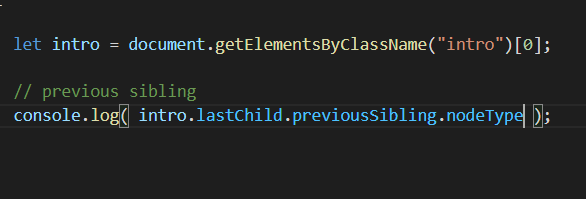




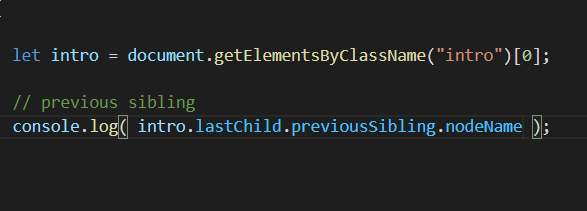


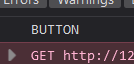






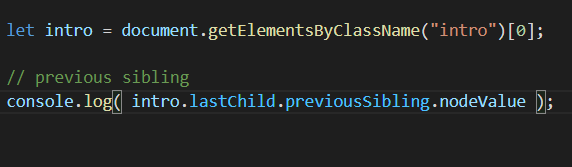


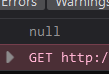




**Be careful**

**accessing nodeValue could be tricky**

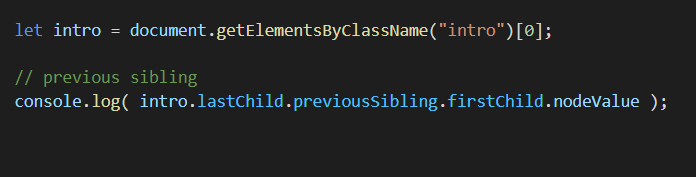




See

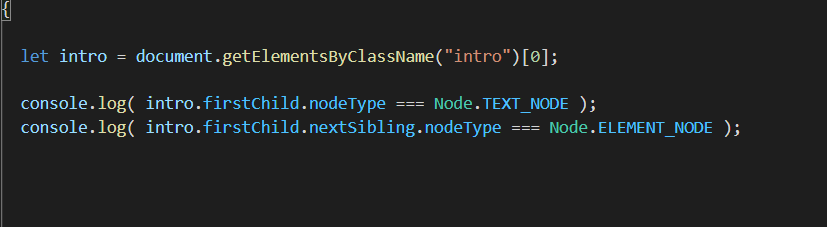
This happened because

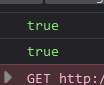
It has another text node inside it and that has the node value that we are looking for





.nodeType and .nodeName properties are useful for conditional cases

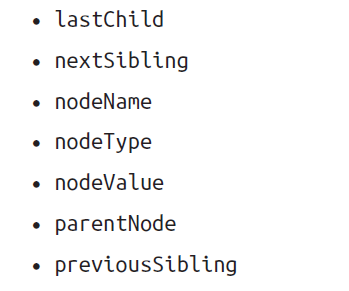




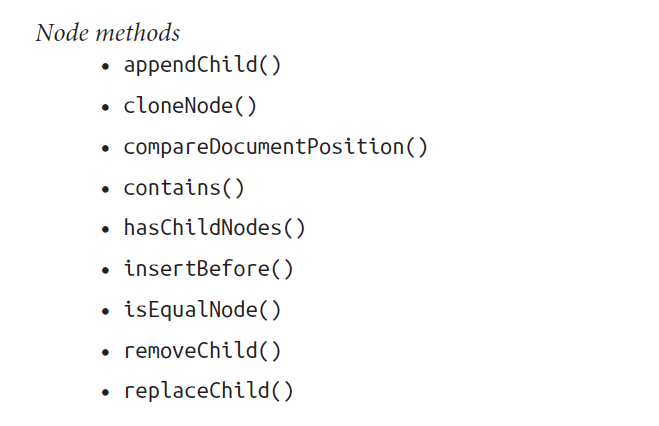
We have provided with more than enough methods and properties to work with and manipulate DOM

These are Node properties





These are Node methods

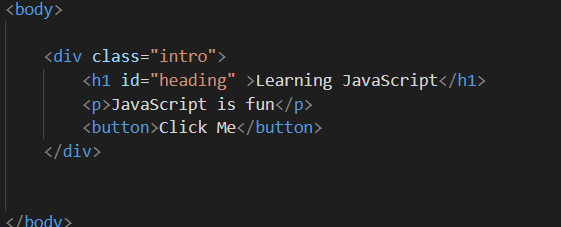


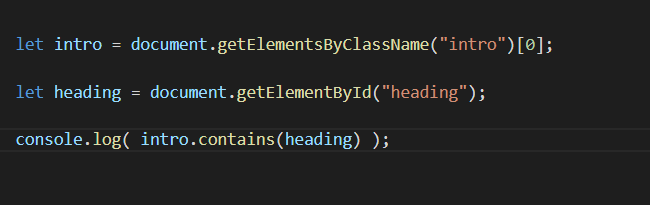
Now we could achieve similar result by easier way by using HTML Element Properties, but certain Node methods are just very handy and useful

**.hasChildNodes()**

returns true or false depending upon the given node has child nodes or not

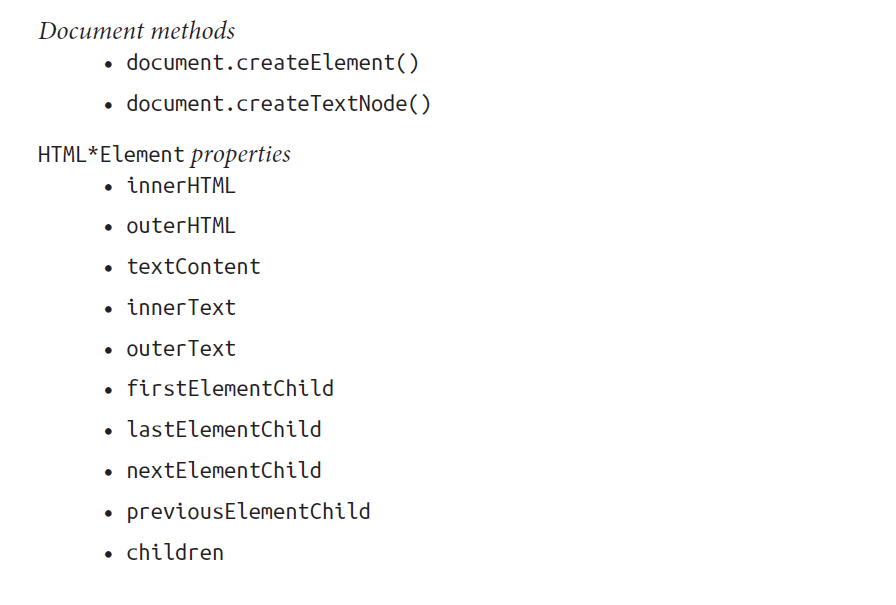
**.contains()**





Returns Boolean depending whether the intro node contains heading node inside it or not

**These are other methods and Properties used for DOM manipulation**

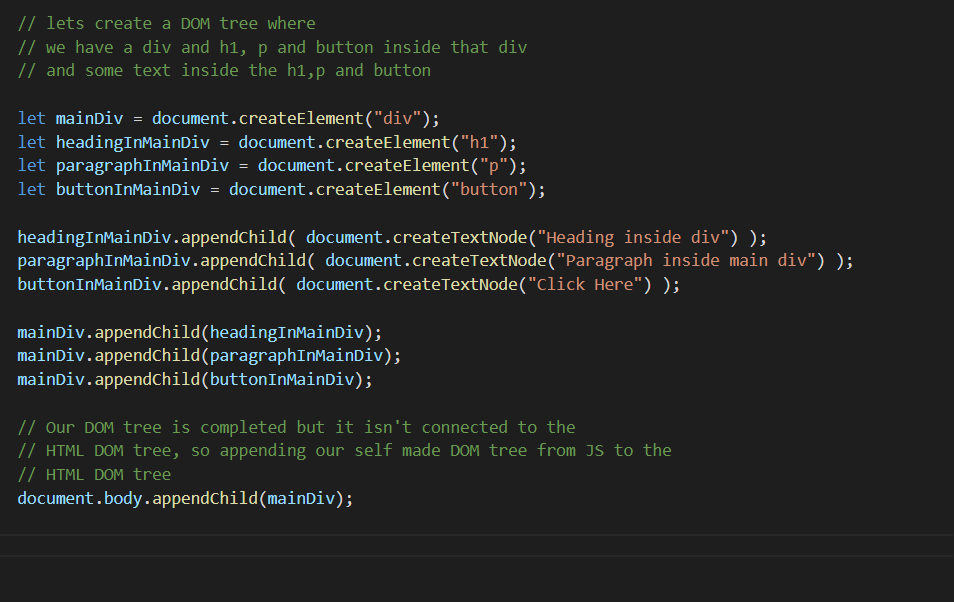


The first two are frequently used and really handy / powerful and useful

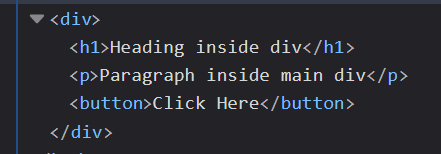
document.createElement() : Used to create any element tag / Node in javascripe

document.createTextNode() : Used to create Text Node in javascript

**Creating a small DOM tree in JS and appending that to the HTML DOM tree. Doing this we will go through various iterative process and see which methods and properties are best to work with**



By this we made a DOM tree that looks like this



First Iteration

instead of .appendChild() we could use .append()

.appendChild() actually works by

variable\_holding\_parent\_node\_object**.appendChild(** variable\_holding\_child\_node\_object **)**

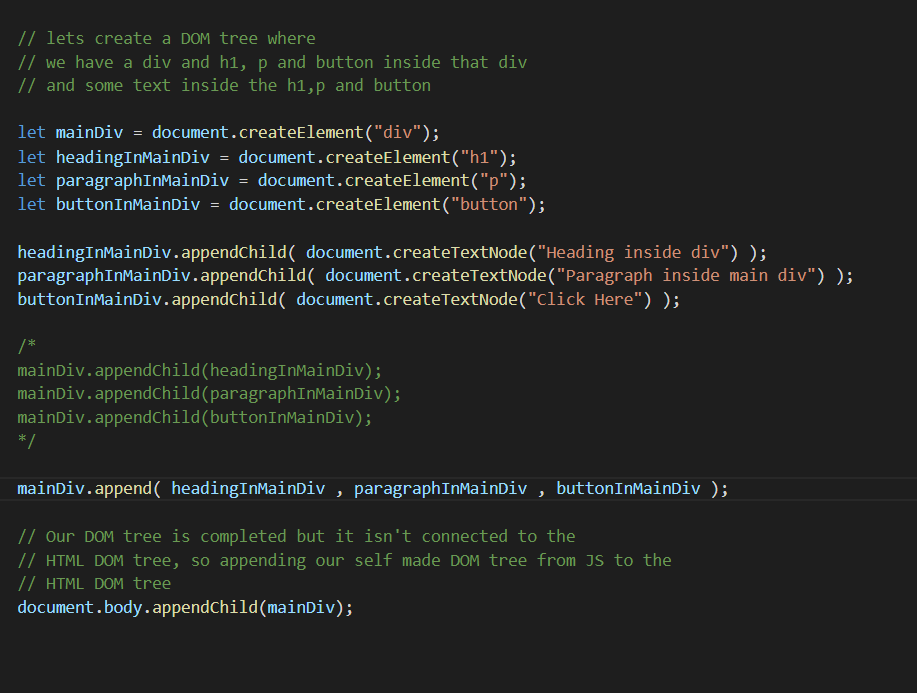
* So here we can append only one child node to a parent node at a time
* Also the child node must be a node object, meaning let say we want to append a string / text inside any element, than first we have to wrap that text inside any text Node and then only we can append using this method. Whereas using .append() method we could append “text\_string” directly without converting that to text node

.append() work by

variable\_holding\_parent\_node\_object**.append(** variable\_holding\_child\_node\_object\_1 , variable\_holding\_child\_node\_object\_1 , variable\_holding\_child\_node\_object\_1 , direct\_text\_string … … … as many number as we want**)**

* So here we can append a node object or “text\_string” directly without converting it to the text node
* We can append more than one node at a time

Using .append() instead of .appendChild() in above code

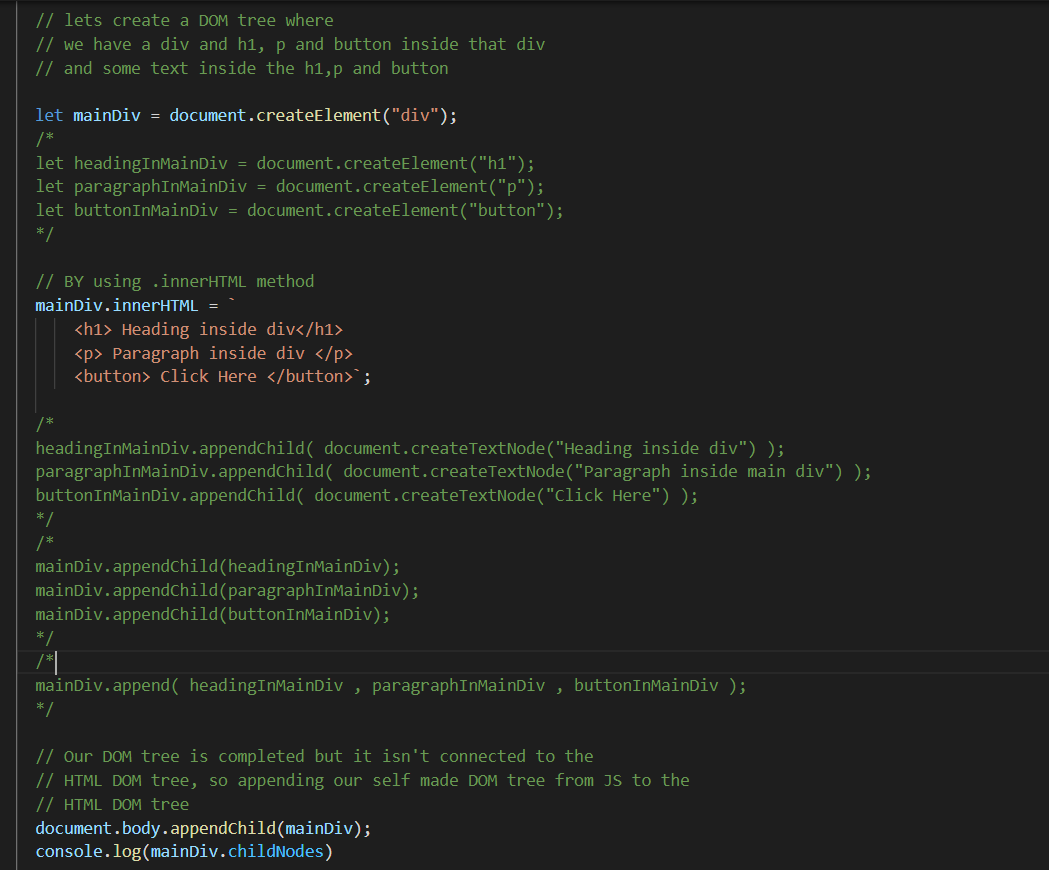


**So using .append() instead of .appendChild() is best choice**

**We have further simpler method**

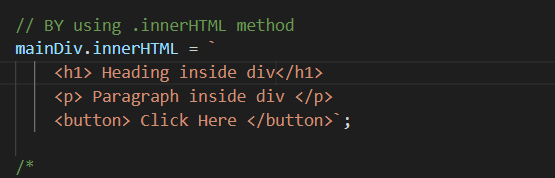
**.innerHTML () method makes things further simpler in constructing DOM in Javascript**

See below



We achieve similar result as above using .innerHTML() which is a lot more easier to read and simpler

Understand how .innerHTML works ?



See what is happening here

* We basically passed string
* than we get the desired DOM tree
* So the .innerHTML is basically looking at our passed text string and parsing the HTML present in it

Why .innerHTML is slower ?

Since it has to parse the passed text string and extract the HTML and construct the DOM tree, so it is a lot more slower.

So use .innerHTML to parse lesser number of HTML and smaller projects

**The Conclusion**

We have two ways of constructing DOM tree by javascript

1. Using innerHTML
2. Using

.createElement() to create a node / Element

.createTextNode() to create a text node

than appending and constructing the DOM

The second one is faster than the first one

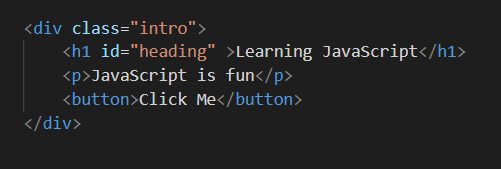
**nodeValue vs textContent**

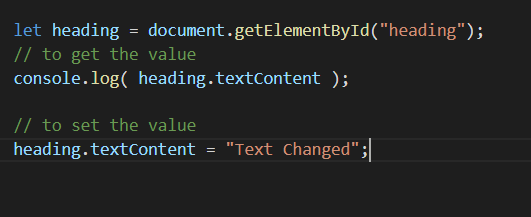
Both are used to get the text value of the node / element as well as to set the value

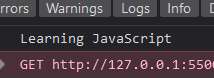
Seems both nodeValue and textContent are similar in performance

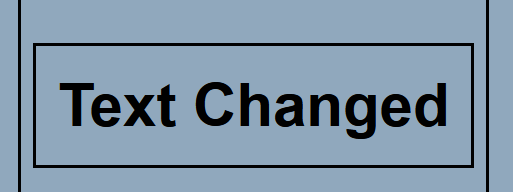
Using textContent than nodeValue is lot more simpler and handy

Using textContent to get the text value of any node / Element

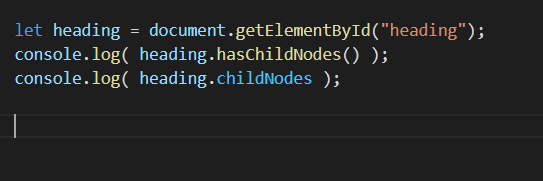


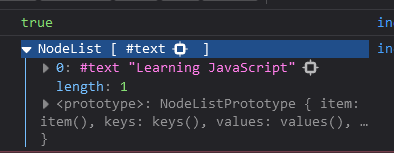






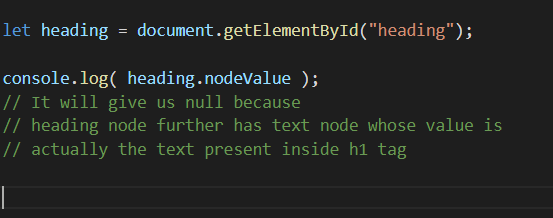
Using NodeValue



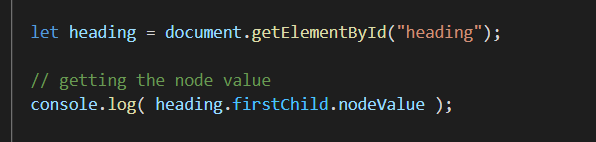


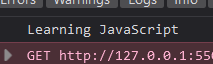
We see heading node further has text node inside it

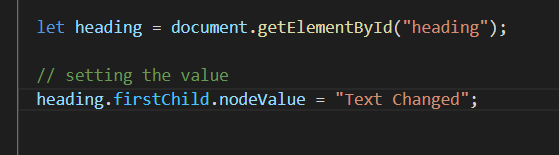
So doing this won’t give us the text inside it

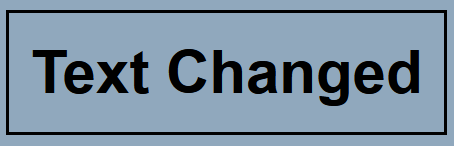


So we have to something like this



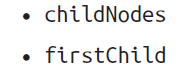


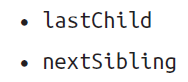


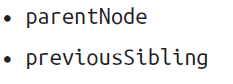


**Further Simpler way to interact with the HTML DOM**

These are the Node properties to interact with HTML DOM





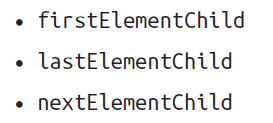


**These are the classical and faster properties to interact with the DOM tree. Where as below given properties are newly introduced and slower compared to these, but very easier and handy than these.**

These are the alternatives of all above properties

These are HTML Element Properties to interact with HTML DOM







Also there is parentElement instead of classical parentNode property

**.childNodes vs .children**

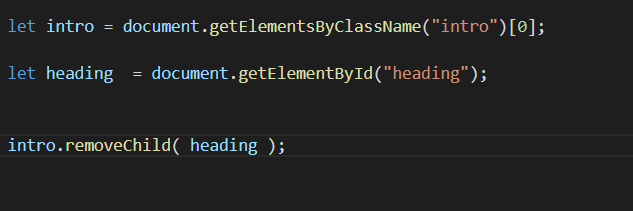
.childNodes() gives us all the nodes present inside the node, it also includes text nodes created by \n (new line), (space) “”

But .children() only gives us the HTML Element nodes present inside the given node object without eliminating the text nodes created by new line and space in it’s node list

**Using firstElementChild , lastElementChild , nextElementChild , previousElementChild instead of firstChild , lastChild , nextSibling and previousSibling**

Using latter methods we may again encounter with the unwanted text nodes created by new line ( \n ), white space ( “” ) . Whereas using first ones we will always get the desired HTML Element nodes

**To delete a certain child node present inside a parent node**

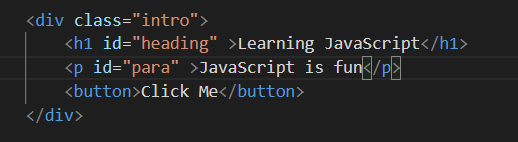


first get both parent element and child element as nodes

Than use method .removeChild( )

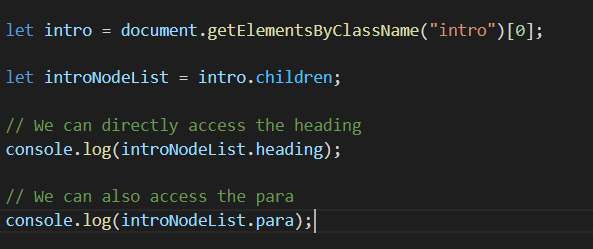
**There is one more advantage of using .children to get the HTML Collection instead of using .childNode to get the nodelist has one very handy and important advantage**

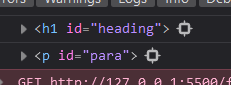
By doing so we can directly access those HTML elements ( having IDs in them ), by treating the IDs as the key of the HTML Collection object ( returned by .children property )



Here inside div, we have h1 having ID “heading” and p having ID “para” .

So here if we get the HTML Collection of div using .children we can access the elements having IDs in them





See

We access the

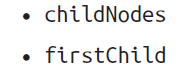
<h1> having “heading” id and <p> having “para” ID

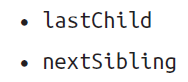
Here  this is kind of wrong variable name

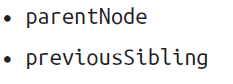
Actually . children property returns us the HTML Collection instead of nodelist. So remember that

Variable naming like this  would have been better

**Best Practice while manipulating DOM tree**



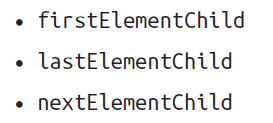




So instead of these

Using these might be easy and simple





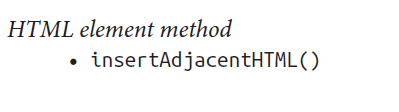


**.innerHTML**

it is really powerful and make things simpler. But it accepts the html elements in the form of string so it has to do a lot of work to parse the string form to actual DOM . And this reduced efficiciency

So limit .innerHTML for short and small html elements insertion

Try avoiding .innerHTML inside the loop, it reduces efficiency



It might be more efficient in comparison to .innerHTML, but at the end this also parsing the passed string in to actual html elements and adding them to the DOM. So this is also not efficient

Various Node methods are also super useful and efficient.

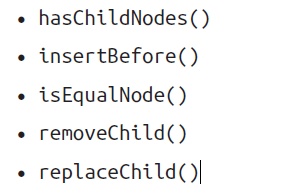
Try using them

like



**.append ( )**

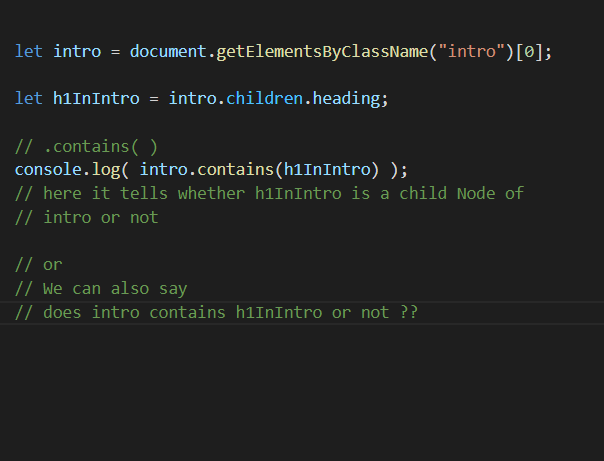
**.contains( )**



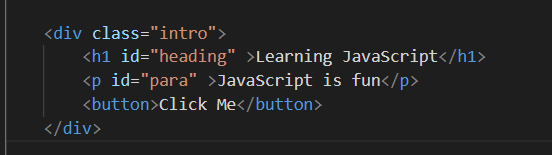
All these are super efficient and easy

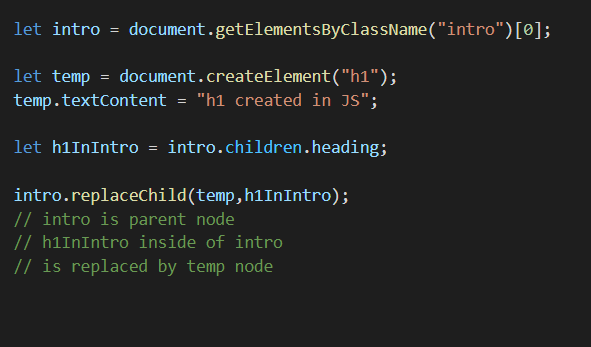
So try using these

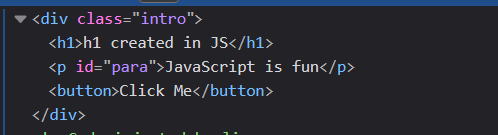
**.contains( )**



**.replaceChild( )**





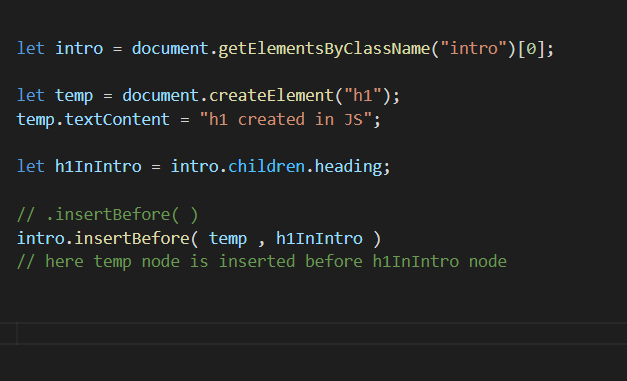


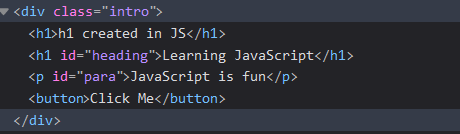


**temp 🡨 h1InIntro**

**so remember it like this h1InIntro node is replaced by temp node**

.insertBefore( )





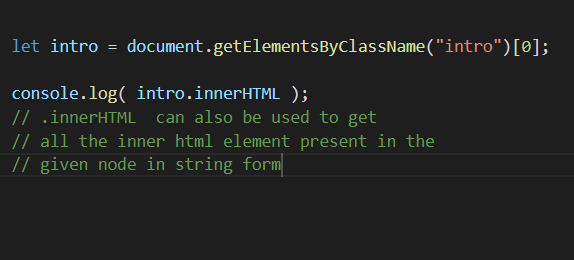
**intro.insertBefore(temp , h1InIntro)**

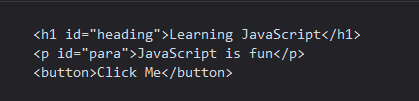
**so remember it like this temp node is inserted before ( aagadi ) h1InIntro node**

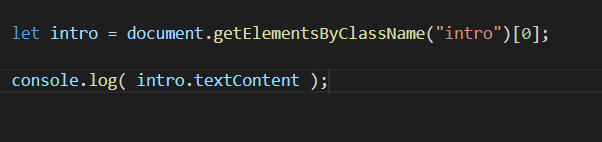
**Well there is another method which we will learn later on which is much more handy and easier than this .insertBefore() one**

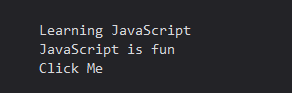
**.innerHTML and .textContent**

we can use these not only to set but we can also get the innerHTML elements in the string form and text inside the element node in string form.









As we can see .textContent will get all the text nodes contained inside the provided element node object element, not just the first text node.

**More useful methods**

**.append( )**

**.prepend( )**

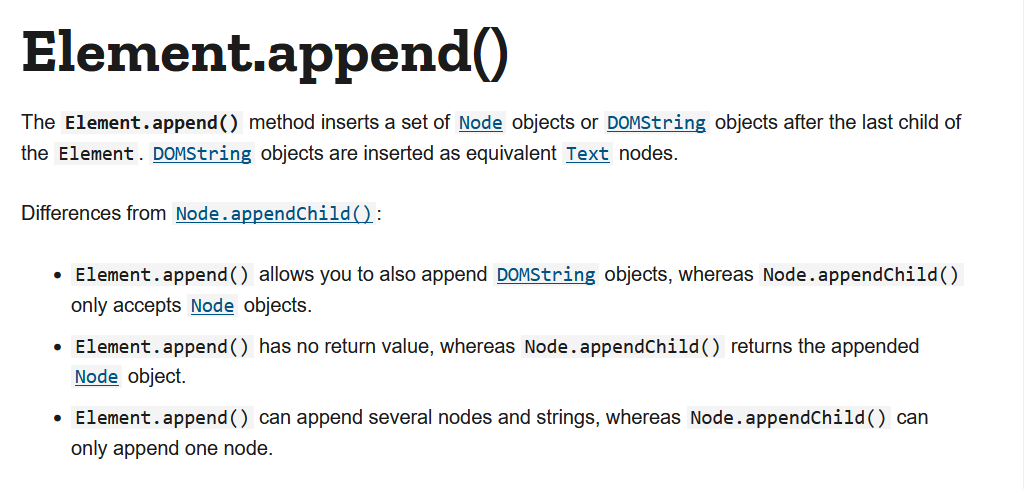
**.before( )**

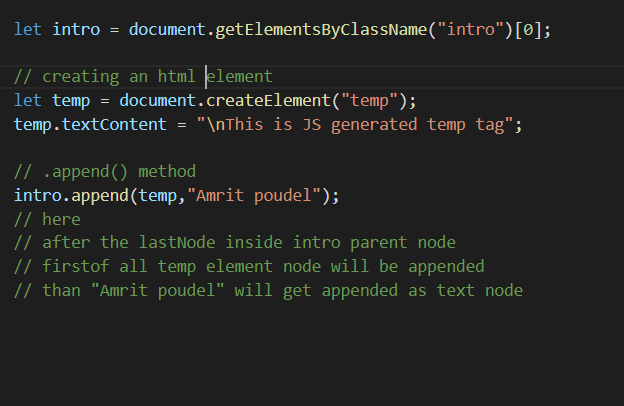
**.after( )**

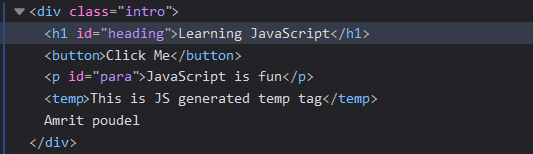
**actually these are introduced in latest DOM 4 to use in place of old methods like .appendChild() , .insertBefore()**

These methods were added in the latest version of the DOM, i.e DOM 4

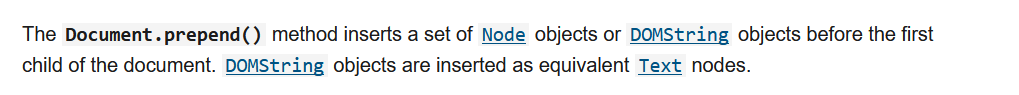
**.append( )**

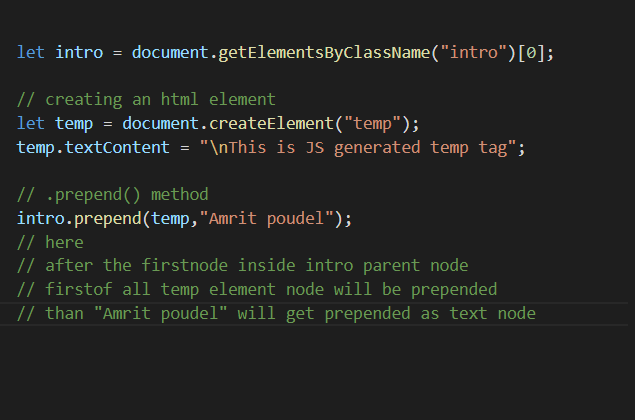


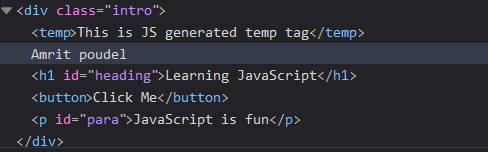




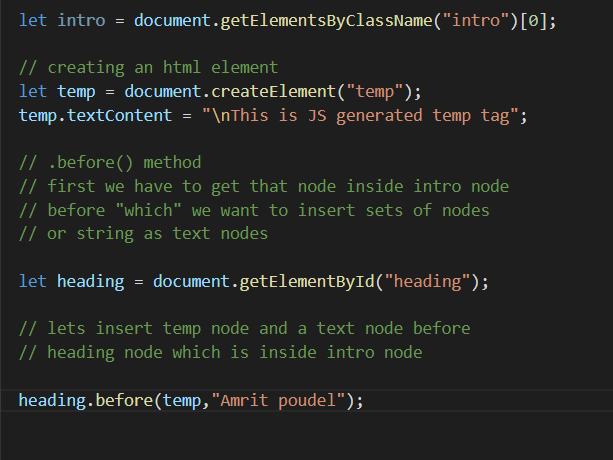
**.prepend()**

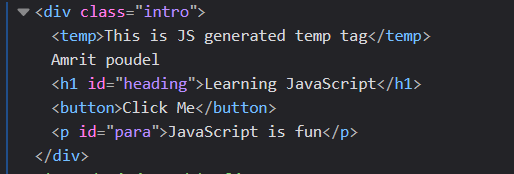




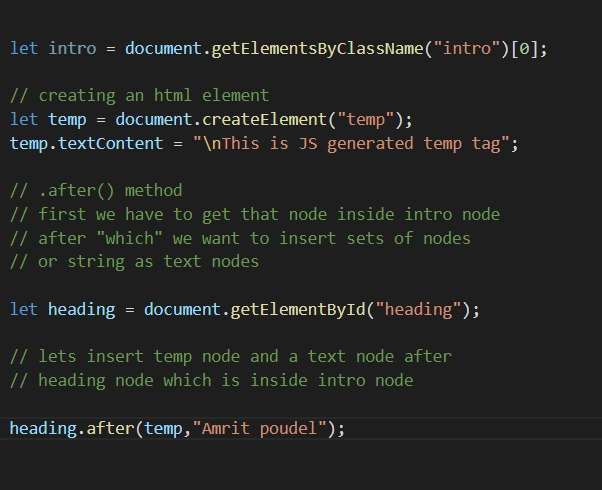


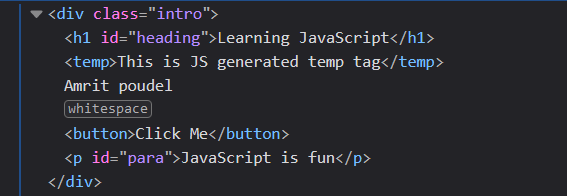
.before( )





.after( )





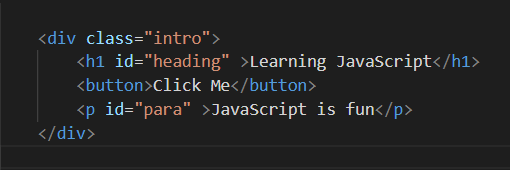
**Another powerful operation in DOM tree**



It may not seem but to be able to remove a node or replace with another is really powerful

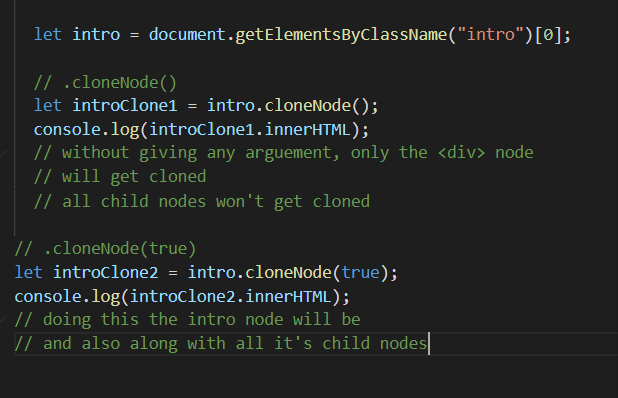


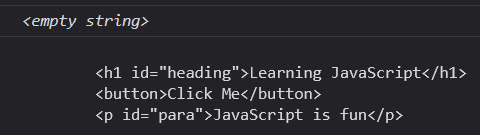
This one is really useful and handy to clone any node. Also it is really fast than any other ways of cloning any node.

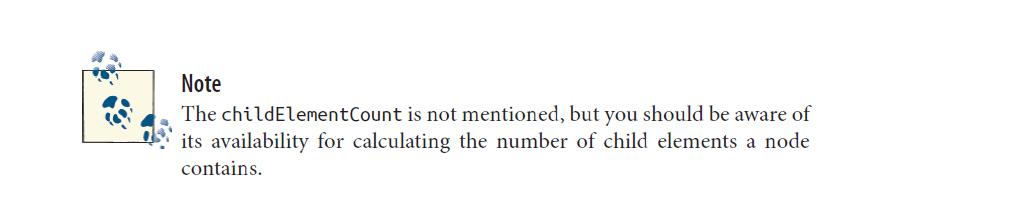


Let say we wanna clone div.intro node

.cloneNode() actually takes argument

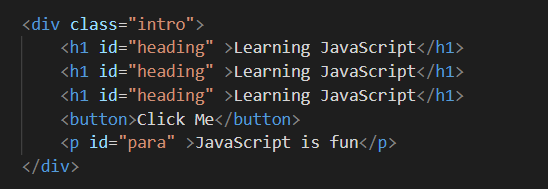






Very handy and useful

Counts the number of elements present inside the parent element object / parent node object



These are 5 elements inside <div>element

