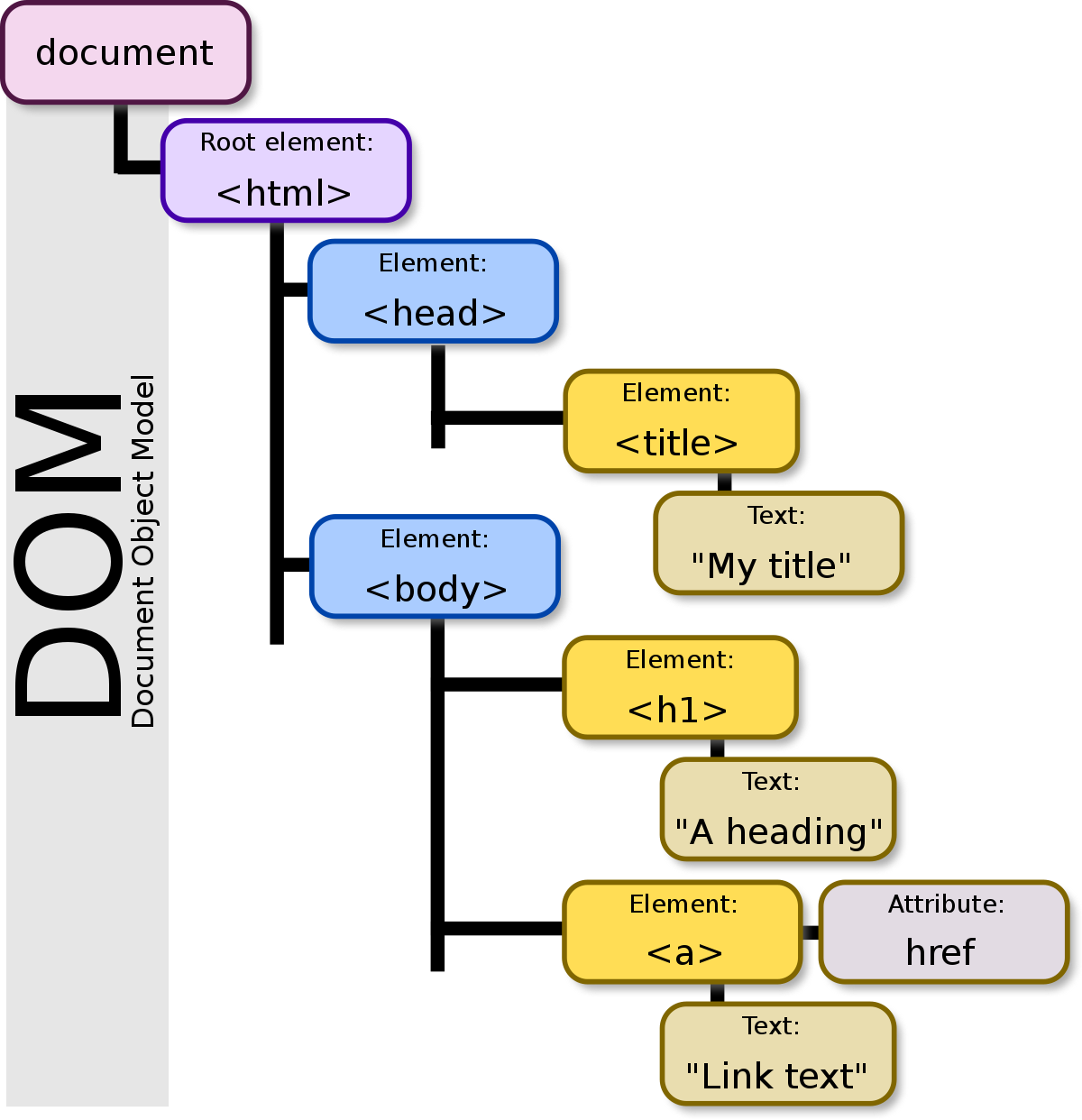
# **DOM (Document object model)**

HTML dokumento kaip Javascript objektas modelis

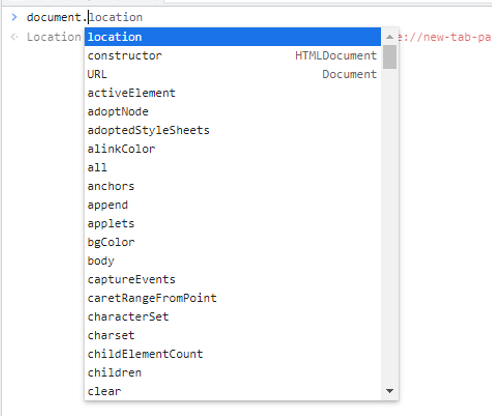
## Kas yra DOM?

Kuomet web puslapis yra užkraunamas, interneto naršyklė sukuria html dokumenta kaip objektą, kurį sudaro kiti elementai kaip objektai taip sudarant HTML elementų objektų medį:



Pagrindinis objektas vadinamas document. Jis savyje kaip objekto atributus laiko visus reikalingus metodus ir savybes norint manipuliuoti visu html medžiu.

Pačių savybių ir metodų yra labai daug, todėl jas visas mokėti tikrai nereikia, apžvelgsime tik pagrindinias. Bet jei norite žinoti kokios yra pasiekiamos savybės ir metodai, galite interneto naršyklės konsolėje, tiesiog įvesti document ir uždėjus tašką, naršyklė jums parodys kas yra prieinama.



Visa šitą HTML medį įsivaizduokite kaip didžiulį jums pateiktą JavaScript objektą kurį jūs galite modifikuoti taip kaip modifikuojami visi Javascript objektai. DOM buvo sukurtas kaip naršyklės dokumento modifikavimo interface’as, kad Javascript pagalbą galėtume padaryti html pakeitimus, taip sukurdami animacijas, sekti vartotoju įvedimus, veiksmus ir daugybę kitų dalykų.

Interneto naršyklė taip pat pateikia priėjimą prie kitų naršyklės interface’ų. Vienas iš jų yra globalus objektas window. Tai nėra DOM dalis, nors veikia panašiu principu. window objektas laikomas kaip šakninis ir pradinis Javascript objektas, kuris savyje turi visus Javascript elementus interneto naršyklėje. Visi interneto naršyklės interface’ai yra aprašyti čia:

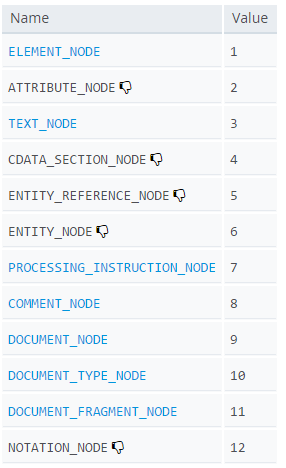
<https://developer.mozilla.org/en-US/docs/Web/API>

Kaip matome jų yra tikrai daug, ir kaip pvz document ar window, yra tiesiog našyklės interface’ai, kurie suteikia programuotojui kontrolę atitinkamų kompiuterio ar pačios naršyklės dalių. Per šiuos interface’us mes galime gauti reikalinga informacija apie kompiuterį ar vartotoją, kaip pvz [Geolocation API](https://developer.mozilla.org/en-US/docs/Web/API/Geolocation_API) ir galime daryti pakeitimus, bei duoti interneto naršyklei komandas. Visi šie interface’ai yra Javascript interface’ai, o tai reiškia, kad jie yra pasiekiame per Javascript koda.

## Nodes vs Elements

Kuomet suformuojamas dokumento objekto modelis (DOM) jis yra išskaidomas į node’us. Node yra bendrinis vardas kiekvieno DOM objekto hierarchija. Node’u gali būti vienas iš vietinių DOM elementų kaip document ar document.body, tai gali būti HTML tag’as, kaip po <p> ar <h1>, galimas teksto node’as, kuris sukuriamas automatiškai, kad išlaikytų html kodo formatą (visos baltos ir tuščios vietos, yra text node’ai). Trumpai tariant Node’ai yra kiekvienas DOM objektas.

Elementas yra tiesiog konkretus node tipas. Žemiau esančioje lentelėje matome galimus Node tipus.



DOM node’ai yra sugeneruojami į hierarchinį medį, kuomet kiekvienas iš jų turi savo vaikinius node’us (child nodes), po juo ar prieš jį einantį šeimos narį (nextSibling arba previousSibling) ar tėvinį node’ą. document objektas po savim turi html kaip vaikinį node’ą. html node’as turi lista kitų vaikinių node’ų kaip po head arba body. Body node’as turi savo vaikus ir kartojasi, iki kol yra išlistintas kiekvienas sugeneruotas node’as DOM medyje.

**Kokius veiksmus galima atlikti su DOM Javascript interface’u?**

* Mes galime keisti visus HTML elementus esančius puslapyje.
* Mes galime keisti visus HTML tag atributus puslapyje.
* Mes galime keisti CSS stilius.
* Mes galime ištrinti egzistuojant HTML elementą ar atributą.
* Mes galime sukurti ir pridėti naujus HTML elementus ir atributus.
* Javascript gali sureaguoti į visus egzistuojančius HTML įvykius puslapyje. Tokius kaip paspaudimas pelės mygtuko, ar pelės kursoriaus užvedimo ant atitinkamo taško, klausytis skrolinimo veiksmo, stebėti naršyklės lango padidėjimą ar sumažėjimą.
* Mes galime gauti aktualia informacija, tarkim esamo puslapio ar elemento pozicija x ir y ašyse. Arba elemento plotį ir aukštį ir juos keisti.

## Pasiekti DOM elementus

Norėdami pasiekti HTML elementus, tam galima panaudoti du pagrindinius document objekto metodus tai: querySelector ir querySelectorAll.

| **element = document.querySelector(selectors);** |
| --- |

| **nodeList = document.querySelectorAll(selectors);** |
| --- |

Abu metodus galima šaukti nebūtinai ant **document** objekto, galima šaukti ir ant elemento node’o.

| elementList = parentNode.querySelectorAll(selectors); |
| --- |

Jei tarkime pirmu gavimu gavome elekemtnus, ir vėliau norime gauti tuose elementuose priklausančius kitus paveiksliukus, tai galetume daryti šitaip:

| const item = document.querySelector('.card'); const itemImages = item.querySelectorAll('img'); |
| --- |

Atkreipkite dėmesį, kad querySelector ir querySelectorAll argumentai yra CSS selektoriai. Tai reiškias kad jie veiks tokiu pačiu principu, taip kaip jūs nurodote kuriems elementams uždėti stilius rašydami CSS. Pvz. document.querySelector('.card') ieškos pirmo elemento, kuris turi css klasę “card”.

Metodas **querySelector** grąžina pirma elementą atitikusį parašyta argumentą, **querySelectorAll** grąžina lista node’ų kurie atitiko pateikta argumentą. nodeList Tai yra panašus duomenų tipas kaip masyvas (array), tačiau turi limituotas galimybes lyginant su originaliu Javascript masyvu.

Jeigu mums reikia masyvo atributų, mes nesunkiai galime transformuoti nodeList į array su Array pagalbiniu metodu FROM. Array.from(nodeList).

(pastaba: senesnės naršyklės kaip IE12 ar kt. nepalaiko šio metodo, todėl jam reikia parašyti polyfill’a);

Yra ir kitų document metodų, kurie parenka HTML elementus:

| document.getElementById() document.getElementsByClassName() document.getElementsByName() document.getElementsByTagName() |
| --- |

Apie juos daugiau galite pasiskaityti Web Docs.

<https://developer.mozilla.org/en-US/docs/Web/API/Document/getElementById>

<https://developer.mozilla.org/en-US/docs/Web/API/Document/getElementsByClassName>

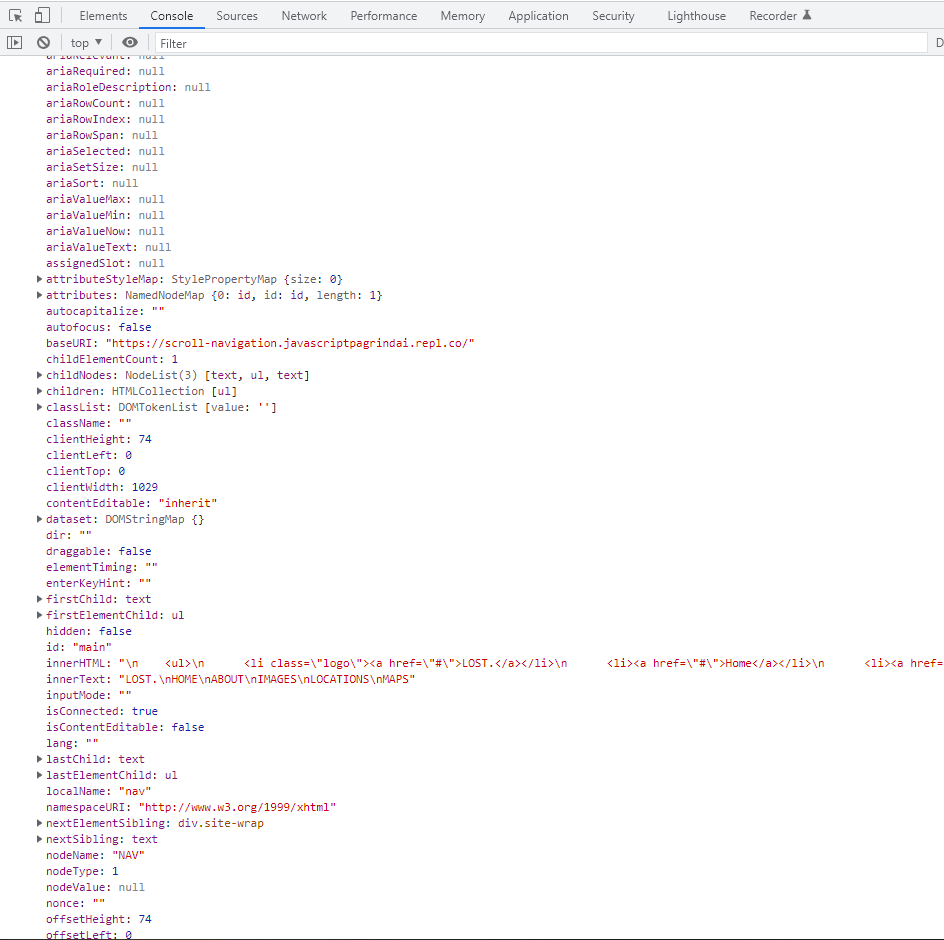
<https://developer.mozilla.org/en-US/docs/Web/API/Document/getElementsByName>

<https://developer.mozilla.org/en-US/docs/Web/API/Document/getElementsByTagName>

## Elementų savybės ir metodai (properties and methods)

Pasirinke elementą su **querySelector** mums yra grąžinamas HTML elemento node’as kaip Javascript objektas kuris savyje turi daugybę savybių. Pamėginkite konsolėje pasirinkti bet kokį elementą ir išspausdinti į ekraną.

| **element = document.querySelector('img'); console.dir(element);** |
| --- |

****

Kaip matome šis elementas turi daugybę savybių ir metodų savyje. Iš jų 95% mums nėra reikalingi ar dažnai naudojami, tačiau jei prireikia kažkokios konkrečios savybės atitinkamam tikslui pasiekti, tai visada galime išspausdinti elementa su console.dir(element) ir matyti visas savybes kurias tas elementas turi.

Daugiau informacijo apie visas elementų savybes rasite šioje nuorodoje:

<https://developer.mozilla.org/en-US/docs/Web/API/Element>

**Elemento metodai tai yra veiksmai kuriuos galima atlikti ant HTML elemento.**

**Elemento savybės yra reikšmės to HTML elemento, kurias mes galime pasiekti ir tuos duomenimis panaudoti, arba juos perrašyti.**

Pavyzdys jei norėtume gauti paragrafo tekstą.

| element = document.querySelector('p.text'); *// Pasirenkame paragrafą su css klase ‘text’.*  console.log(element.textContent); *// Teksta pasiekiame su savybe textContent per taško notacija.* |
| --- |

Pavyzdys jei norėtumėte gauti paragrafą supantį html su pačiu paragrafu.

| *element = document.querySelector('p.text'); // Pasirenkame paragrafą su css klase 'text'.  console.log(element.outerHTML); // Gauname HTML pasiekę per outerHTML.* |
| --- |

Pavyzdys jei norėtumėte gauti div elementa su jame esančiu html.

| *element = document.querySelector('div.card'); // Pasirenkame div su css klase 'card'.  console.log(element.innerHTML); // Gauname HTML pasiekę per innerHTML.* |
| --- |

Pavyzdys jeigu norėtumėte įdėti teksta prie jau egzistuojančio teksto

| *element = document.querySelector('p.text'); element.insertAdjacentText('beforebegin', '🍕'); /\* insertAdjacentText priima keturis pirmo parametro tipus, priklausomai kur norite įdėti tekstą. 'beforebegin' 'afterbegin' 'beforeend' 'afterend'  Antra parametras yra tekstas, kurį norite įdėti.   \*/* |
| --- |

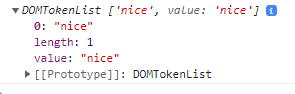
Taip pat galima padaryti su HTML ir Elementu. **element.insertAdjacentHTML()**, **element.insertAdjacentElement()**.

## HTML elementų CSS klasių modifikavimas su DOM

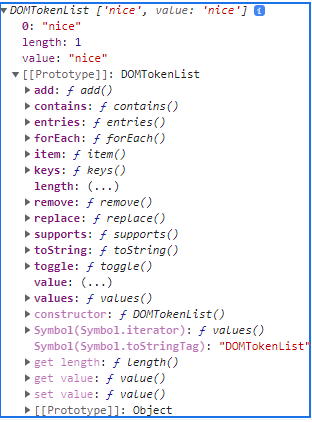
Darbui su CSS klasėmis galime panaudoti elemento savybę “classList”, kuris yra irgi specializuotas objektas DOMTokenList turinti reikiamus metodus manipuliuoti css klases.

| <html> <body>  <div id="main">  <img class="nice" src="https://picsum.photos/600" alt="">  </div>   <script>  const pic = document.querySelector('.nice');  console.log(pic.classList)  </script> </body> </html> |
| --- |
|  |

Paleide aukščiau esantį koda, pamatysite, kad consolė mums gražina DOMTokenList.



Praskleidus DOMTokenList prototipa matome, kad jis savyje turi labai naudingų metodų:



Pagrindiniai naudojami:

**add()** - pridėti css klasę

**remove()** - pašalinti css klasę

**toggle()** - pašalina klasę jei ji jau egzistuoja, prideda klasę jei jos tuo metu nėra.

**replace()** - pakeičia egzistuojančia klasę, jei ji yra.

| const pic = document.querySelector('.nice'); pic.classList.add('image'); *// Pridedam css klase "image"* pic.classList.remove('image'); *// Ištrinam css klase "image"* pic.classList.replace('nice', 'very-nice'); *// Pakeitėme "nice" klase i "very-nice"* |
| --- |

## HTML atributų modifikavimas

Kadangi html elementai yra kaip objektai, tai mes juos galime keisti tiesiog priskirdami naują reikšmę, kadangi jie yra “getters” and “setters”.

Pvz:

| const pic = document.querySelector('.nice'); pic.alt = "nice image"; pic.width = 200; |
| --- |

Rezultatas:



**Priskirti arba keisti atributus galima su tam skirtais metodais:**

**getAttribute()** - gražina nurodyto atributo reikšmę/

**getAttributeNames()** - gražina masyva su visais html elemento atributais.

**setAttribute()** - priskiria nurodytam atributui reikšmę.

Šiuo metu šitie metodai yra naudojami mažiau, todėl nerekomenduoju naudoti.

*Pastaba:* setAttribute metodu, gali priskirti savo unikalius atributus, tačiau to daryti, labai nerekomenduojama! Geriau naudokite jau esančius standartizuotus HTML5 atributus.

Pvz:

| const pic = document.querySelector('.nice'); pic.setAttribute('js-cool', 'oh yes'); |
| --- |

Rezultatas:



Tam geriau yra naudojamas **“data-”** atributas, kuriam galite priskirti savo pasirinktą reikšmę.

Naudojant “dataset” atributa, kuris gražina visus data atributus.

Pvz:

| <li><a data-name="home-link" href="#">Home</a></li> <li><a data-selec="home-link" data-event="click" href="#">About</a></li> <li><a data-name="home-link" href="#">Images</a></li> <li><a data-name="home-link" href="#">Locations</a></li> <li><a data-name="home-link" href="#">Maps</a></li> |
| --- |

Pvz1: Pasirenkam a elements su vienu data atributu ir juos išspausdiname konsolėje

| const link = document.querySelector('#main li a'); console.log(link.dataset) |
| --- |

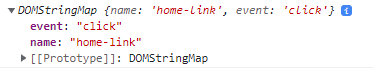
Rezultatas:



Pvz2 : Pasirenkame a elementa su dviem data atributais ir juos išspausdiname konsolėje.

| const link = document.querySelector('#main li:nth-of-type(2) a'); console.log(link.dataset) |
| --- |

Rezultatas:



Pvz3: Pasirenkame konkretų data elementa.

| const link = document.querySelector('#main li:nth-of-type(2) a'); console.log(link.dataset.event) *// click* |
| --- |

## Naujų HTML elementų kūrimas ir pridėjimas į DOM

DOM interface ir javascript pagalba mes galime kurti naujus html elementus ir juos įdėti į HTML medį.

Tam naudojamas document.createElement() metodas.

Pridėti tekstą į naujau sukurta alementa, mes galime pasinaudoti metodu: document.createTextNode()

Pvz: HTML

| <!DOCTYPE html> <html> <head>  <title>||Working with elements||</title> </head> <body>  <div id="div1">The text above has been created dynamically.</div> </body> </html> |
| --- |

Pvz: Javascript

| *// Sukuriame nauja div elementą* const newDiv = document.createElement("div");  *// Priskiriame div elementui turinį* const newContent = document.createTextNode("I love Javascript");  *// Pridedame text node'a į div elementa* newDiv.appendChild(newContent);  *// Įkeliame sukurta pilna elementa į DOM* const currentDiv = document.getElementById("div1"); document.body.insertBefore(newDiv, currentDiv); |
| --- |

Pridėti naujus elementus į DOM dar galime panaudoti **insertAdjacentElement** metoda.

targetElement.insertAdjacentElement(position, element);

| */\* Position reikšmės:  'beforebegin' 'afterbegin' 'beforeend' 'afterend' \*/* |
| --- |

### **InnerHTML atributas**

Kurti naujus HTML elementus arba blokus galima ir kaip string. Tam naudojama **innerHTML** elemento atributas.

Pvz.

| const imgWrapper = document.querySelector('.image-wrapper'); const imageHtml = `<img class="nice" src="https://picsum.photos/600" alt="">`; imgWrapper.innerHTML = html; |
| --- |

Mes netgi galime naudoti string interpolation ir įdėti atitinkamus atributus kaip kintamuosius. Taip pat galime įdėti ir visa html bloką.

| const imgWrapper = document.querySelector('.image-wrapper'); const src = 'https://picsum.photos/600'; const alt = 'Cute image'; const cssClasses = 'nice image' const imgWidth = 200; const html = `  <div class='image-container'>  <img class="${cssClasses}" src="${src}" alt="${alt} width="${imgWidth}">  </div> ` imgWrapper.innerHTML = html; |
| --- |

Kitas būdas yra naudojant createRange() ir createContextualFragment() metodus.

Pvz:

| const src = 'https://picsum.photos/600'; const alt = 'Cute image'; const cssClasses = 'nice image' const imgWidth = 200; const html = `  <div class='image-wrapper'>  <img class="${cssClasses}" src="${src}" alt="${alt} width="${imgWidth}">  </div> ` const myFragment = document.createRange().createContextualFragment(html); const imageContainer = document.querySelector('.image-container'); imageContainer.appendChild(myFragment); |
| --- |

**Jeigu jūs gaunate kažkokia tai dalį duomenų kurių jūs nevaldote, tarkim iš vartotojo input, arba iš duombazės, tuomet šiuos metodus innerHTML ir createRange naudokite labai atsargiai!!! Todėl, kad yra galimybė injectinti fiktyvų html su javascriptu ir taip nulaužti jūsų puslapį. Šis procesas vadinamas XSS, cross site scripting.**

## Praėjimas pro elementų Node’us (traversing) arba node ištrynimas.

Kiekvienas DOM elementas yra susietas giminystės ryšiais, kurie atitinkamai sujungia visų elementų medį tarpusavyje. Panaudojus atitinkamas node atributus, mes galime keliauti medžiu tiek nuo pradinio (root) į apačią iki paskutinių vaikaičių node’ų, arba atvirkščiai, keliauti iš apačios į viršų. DOM medį galime įsivaizduoti kaip giminės medį, ir veikiantį pagal ta patį giminės ryšių modelį.

To pasekoje kiekvienas DOM node’as turi atributus:

| element.children element.firstElementChild element.lastElementChild element.previousElementSibling element.nextElementSibling element.parentElement |
| --- |

Pasiekus atitinkamus elementus, mes galime keliauti visu medžiu ir atlikti titinkamas operacijas. Šie atributai labai naudingi norint padaryti “Web Crawler” kuris analizuoja html’a ir renka naudinga informacija.

Norint keliauti per visus elementus, tam labai puikiai gali pasitarnauti rekursija. Rekursija yra tuomet kada funkcija šaukia pati save. Javascript leidžia pašaukti funkcijai iš tos pačios funkcijos vidaus.

Daugiau info: <https://developer.mozilla.org/en-US/docs/Glossary/Recursion>

Pvz: funkcijos kuri traversina visus elemento tėvinius elementus.

| function printParents(el) {  console.log(el);  if (el.parentElement) {  printParents(el.parentElement);  } } |
| --- |

Pvz: funkcijos kuri traversina visus elemento pirmus vaikus.

| function printChild(el) {  console.log(el);  if (el.childNodes[0]) {  printParents(el.parentElement);  } } |
| --- |

Tokiu pačiu principu, mes galime eiti per visa HTML medį ir jei elementas atitinkam mūsų nurodyta salyga jį ištrinti.

Tam naudojamas **remove**() metodas.

| const link = document.querySelector('a.nice'); link.remove(); // Elementas buvo ištrintas iš DOM |
| --- |