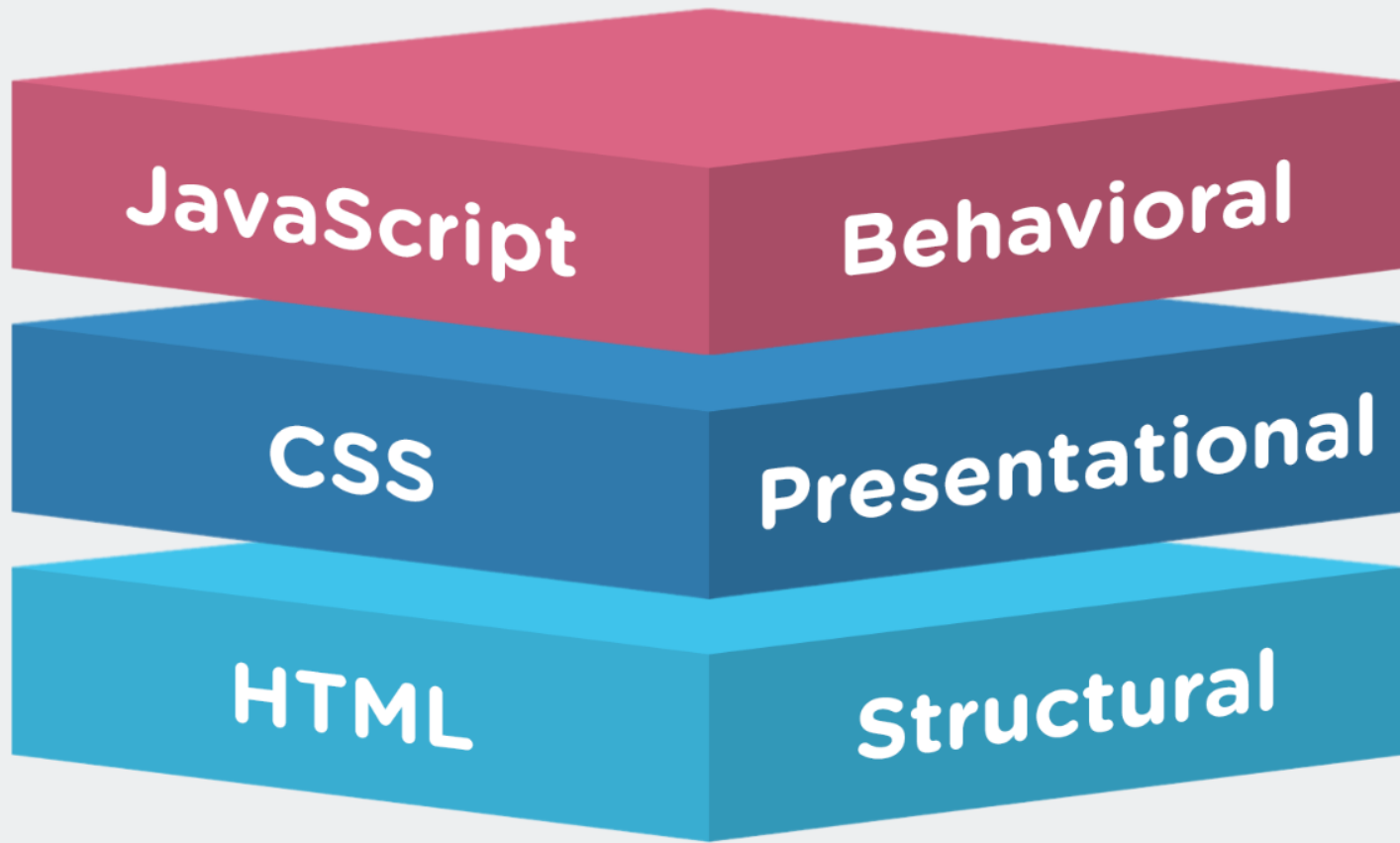




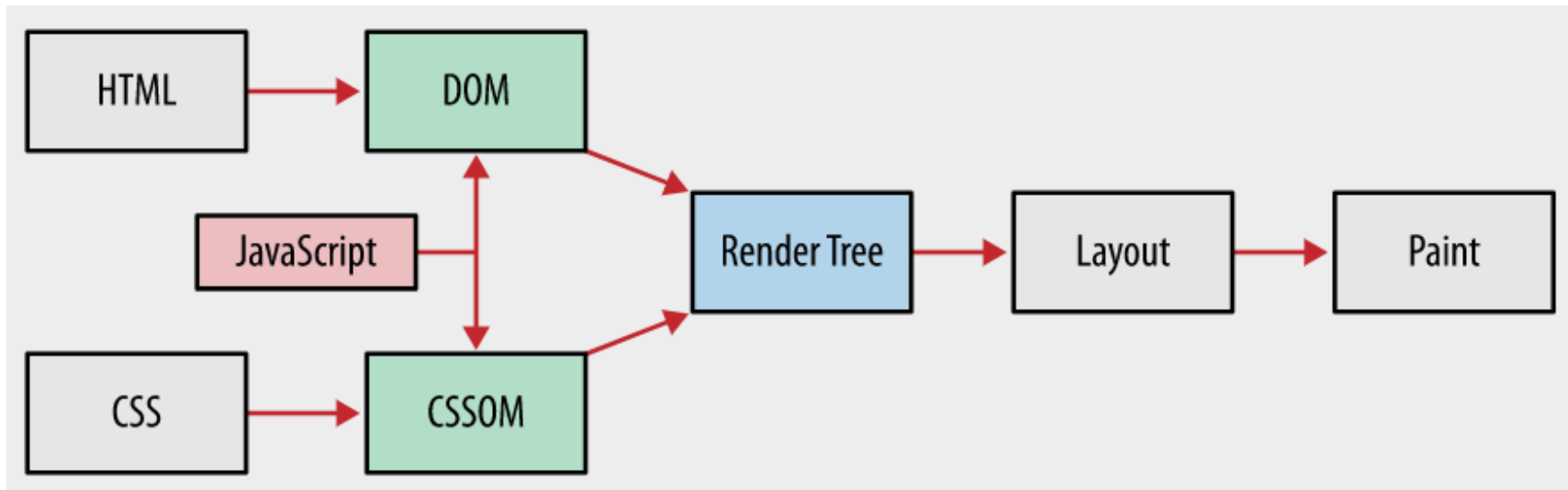
Saptamana 9

Partea 1

Programare Front-End



Ciclul de viata al unei pagini web (*lifecycle*)



1. Working with the DOM

DOM Manipulation

JS

1.1 What about DOM ?

DOM (< Cursul 2)

DOM sau **Document Object Model** este o structura arborescenta alcatuita din obiecte, asociata unei pagini HTML.

Aceasta structura este creata de catre browser in momentul in care o pagina este incarcata, asociind fiecarui element HTML specificat cate un obiect si serveste ca o **interfata** pentru accesarea, modificarea si stergerea elementelor. (vom vedea mai tarziu cine foloseste aceasta interfata)

DOM

- Orice *website* are la baza un document HTML **index.html** pentru a reda structura initiala a *site*-ului
- Browserul creeaza aceasta structura numita **Document Object Model** prin intermediul careia *JavaScript*-ul poate **accesa, manipula, altera** elementele *website*-ului.
- Orice website are propriul DOM creat de catre *browser*-ul in care pagina HTML este deschisa
- Pentru a inspecta elementele existente in cadrul DOM si stilurile aplicate acestora (daca este cazul), putem folosi consola pentru *debugging* pusa la dispozitie de catre *browser*-e

Chrome: Press F12 -> Go to Elements Tab

Exercise: Open Chrome DevTools and inspect the HTML structure and the applied styles for a HTML page

DOM

- DOM - obiect, interfata - prin intermediul sau sunt puse la dispozitie o varietate de metode si proprietati care pot fi accesate, modificate
- Cu ajutorul JavaScript-ului ne putem folosi de aceste metode si proprietati pentru a oferi interactivitate aplicatiilor web (*no more static pages*) - actualizarea paginii fara *refresh*, modificarea stilurilor in functie de actiunile utilizatorului, functionalitati si animatii mai complexe precum elemente *drag-and-drop*, etc... -

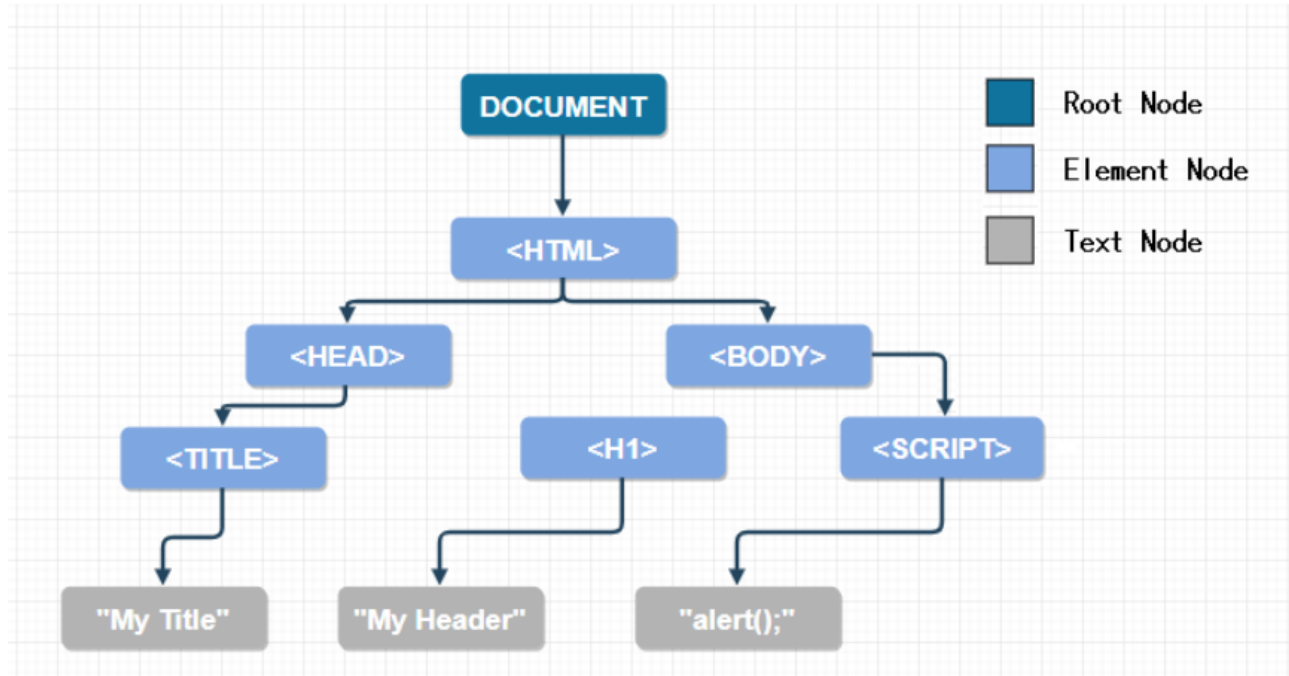
DOM

- EX: `document.body.style.backgroundColor = "orange"` - vom accesa direct DOM-ul
 - `document` este un obiect
 - `body` este o proprietate pe care am ales sa o modificam
 - `style` este atributul prin care stilizam
 - `backgroundColor` este ceea ce modificam iar `orange` culoarea data.

Atentie: Dupa cum se observa, in cod se specifica `backgroundColor` si nu `background-color` - Proprietatile accesibile din Javascript sunt definite si se scriu folosind *camelCase*, fiind diferite de cele scrise cu *snake-case* in CSS

1.2 DOM Structure

DOM Tree & Nodes



DOM Tree & Nodes

- Arborele (*tree*) este alcatuit din obiecte numite noduri (*nodes*)
- exista mai multe tipuri de *nodes* dar vom interactiona cel mai des cu *element nodes* (*html elements*), *text nodes* (*orice text*) si *comment nodes* (*cod comentat*)
- *Nodes* se mai refera si la parinti, copii, frati (*parents, children, siblings*)
- In exemplul alaturat avem parent node - **documentul html**, **head** si **body** sunt **siblings** iar **body** contine 3 nodes care si ei la randul lor sunt **siblings**

```
<!DOCTYPE html>
<html>
  <head>
    <title>Nodes</title>
  </head>
  <body>
    <h1>This is an element node</h1>
    <!-- This is a comment node -->
    This is a text node.
  </body>
</html>
```

Node Types

- Fiecare **node** are un tip (**type**) care poate fi inspectat cu ajutorul proprietatii **nodeType**.
 - [node type list](#): lista cu toate tipurile de noduri
- Daca ne uitam la exemplul din slideul anterior vom vedea ca **html**, **body**, **title** si **h1** sunt de tipul **ELEMENT_NODE** cu valoarea 1, textul este **TEXT_NODE** si are valoarea 3, comentariul este **COMMENT_NODE** si are valoarea 8

Node Types

- Putem verifica unui **node** prin 2 modalitati:

1. Deschidem consola, mergem la *Elements* si inspectam un element din structura HTML: vom vedea la final

`==$0`

- Daca scriem in *Console* `$0` apoi apasam *enter*, ne va aparea **node**-ul pe care l-am selectat in *Elements*.
- Daca scriem `$0.nodeType` ne va aparea o valoare precum `1, 3, 8` in functie de ce am selectat

2. Folosind `document.body.nodeType`

- Mai putem folosi `nodeValue` sa luam valoarea unui text sau `nodeName` sa accesam *tag-name*-ul unui element

1.3 Using DOM Interface

Accesarea elementelor din DOM

- Cu ajutorul urmatoarelor metode putem accesa caracteristici, copii ale elementelor:
 - `getElementById()`, `getElementsByClassName()`, `getElementsByTagName()`, `querySelector()` , `querySelectorAll()`
- `document.getElementById()` - acceseaza un node dupa *id* (*HTML id*)
`<div id="test">I'm an ID</div>`
 - Il putem ulterior putem stoca intr-o variabila `const testEl = document.getElementById('test')`
 - Ulterior, folosind `testId.style.backgroundColor = 'red'`, vom schimba *background-ul div-ului* respectiv, spre exemplu

Atentie: Deoarece ID-ul este unic, functia va returna un singur element

Accesarea elementelor din DOM

- `document.getElementsByClassName()` - selecteaza un set de **node-uri** corespunzatoare unei anumite clase
 - le putem ulterior stoca intr-o variabila `testClasses = document.getElementsByClassName('test');`
 - `testClasses` este de aceasta data un array de elemente `HTMLCollection(2) [div.test, div.test]`
 - Ulterior, folosind `testClasses[0].style.backgroundColor = 'red'`, vom schimba *background-ul* primului element care detine clasa specificata, spre exemplu

```
<div class="test">I'm a class</div>  
<div class="test">I'm another class</div>
```

Atentie: Deoarece clasele nu sunt unice, functia va returna un array de elemente

Accesarea elementelor din DOM

Query selectors - `querySelector()` // `querySelectorAll()`

```
<div id="test-query">Use a query selector</div>
```

```
const testQuery = document.querySelector('#test-query')
```

Daca am avea mai multe elemente: atunci cand folosim `querySelector` se returneaza doar primul element

```
<div class="test-query-all">Use query selector ALL</div>
```

```
<div class="test-query-all">Use query selector ALL</div>
```

Pentru ca toate elementele sa fie returnate, folosim:

```
const testQueryALL = document.querySelectorAll('.test-query-all');
```

Cu ajutorul `querySelector`-ilor putem cauta si dupa tag-uri - `document.querySelectorAll('section, article')`

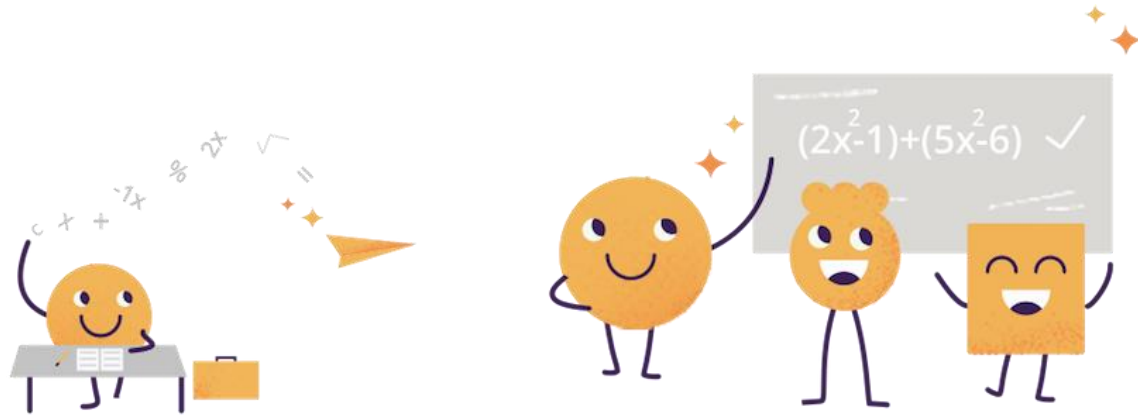
Accesarea elementelor din DOM

- in cazul in care avem mai multe elemente returnate de catre `getElementByClassName()`, acestea pot fi iterate:

```
for (i = 0; i < testClass.length; i++) {  
    testClass[i].style.backgroundColor = 'green';  
}
```

- `getElementsByTagName()`
 - `<section> I am section </section>`
 - vom obtine la fel ca si la clase un *array* - un obiect iterabil

PRACTICE: <https://bit.do/domEx>



1.4 DOM Traversal

Traversarea DOM-ului

- A traversa DOM-ul = a “naviga” prin DOM
- utilizand proprietatile *parent*, *child* si *sibling* putem accesa diferite nivele de noduri - aceste atributii pot fi asociate fiecarui nod in functie de pozitionare, ierarhie, etc...
- **Root Nodes**
 - Document object (document.) este **ROOT**-ul oricarui node din DOM. Mai sus ar fi window object
 - Implicit, fiecare document are html, head si body

```
document.head;    // ► <head>...</head>  
document.body;    // ► <body>...</body>
```

Traversarea DOM-ului

- **Parent Nodes**
 - fiecare nod poate fi parinte in functie de ierarhia din DOM (mai putin **ROOT**)
 - De ex **<html>** poate fi parinte pentru **<body>** si **<script>**
 - **<body>** la randul sau poate fi parinte pentru **<p>** si **<h1>**, spre exemplu
 - putem verifica cine este parintele unui node cu `element.parentNode` (`p.parentNode`, `h1.parentNode`)
 - putem merge si mai deep de exemplu `p.parentNode.parentNode` si va cauta un level mai sus

Traversarea DOM-ului

Child Nodes

- Copii fiecarui node sunt nodurile care se gasesc cu un nivel mai jos (frunze - in arborele DOM)
- Exista mai multe proprietati utilizate in practica pentru accesarea "copiilor": `childNodes`, `firstChild`, `lastChild`, `children`, `firstElementChild`, `lastElementChild`
- <http://bit.do/domTraverse>
 - folosind `ul.childNodes` in exemplu din link vom obtine: `NodeList(7) [text, li, text, li, text, li, text]`
 - *text nodes* sunt defapt spatiile goale cauzate de indentarea dintre elemente iar DOM-ul le interpreteaza ca fiind *nodes*. Nu le putem vedea in *Elements* tab din DevTools deoarece aceasta le scoate automat.
- Atentie: `ul.firstChild.style.background = "red"` - `Uncaught TypeError: Cannot set property 'background' of undefined.`
- Pentru a evita situatia de mai sus vom folosi proprietatile `children`, `firstElementChild` si `lastElementChild`.
- `Ul.children` va returna doar li-urile.

Traversarea DOM-ului

Sibling Nodes

- "Siblings" sunt doua noduri care se afla pe acelasi nivel (*tree level*) in DOM. **Nu trebuie sa fie acelasi nodeType !**
- Proprietati cu care putem lucra :
 - `nextSibling`
 - `previousSibling`
 - `nextElementSibling`
 - `previousElementSibling`

Practice:

- Pentru exemplul din linkul anterior selectati li-ul mijlociu (al doilea) si schimbati culoarea *sibling-ului* anterior si urmator folosind 2 metode de mai sus.

1.4 DOM Updates

Change the DOM!

- Am vazut cum se selecteaza node-urile - cum le modificam, adaugam, stergem ?
- Avem cateva metode precum : `createElement()` si `createTextNode()` cati si proprietati precum `node.textContent` si `node.innerHTML`.

Crearea de noi node-uri:

- `const paragraph =document.createElement('p');`
`//console.log(paragraph) -> <p></p>`
`paragraph.textContent = "hello Wantsome!";`
`//console.log(paragraph) -> <p>hello Wantsome!</p>`
- Combinand cele 2 metode putem obtine un element node complet
- Putem folosi si `paragraph.innerHTML = "Hello mighty! Wantsome!";` pentru a adauga si cod html in paragraful nostru.

```
<!DOCTYPE html>
<html lang="en">

<head>
  <title>Master the DOM!</title>
</head>

<body>
  <h1>Master the DOM!</h1>
</body>

</html>
```

Change the DOM!

Inserarea node-urilor in DOM

- Cu metodele de mai sus am creeat nodes dar inca nu le avem in DOM prin urmare avem nevoie sa le adaugam.
- Avem metode precum : *appendChild()* *insertBefore()* si *replaceChild()*;
- <http://bit.do/changeDOM> - sa folosim impreuna metodele sa vedem ce se intampla.

Stergerea node-urilor din DOM

- Pentru a scoate node-uri din DOM folosim *removeChild()* pentru a sterge un copil(child) din parinte sau putem folosi *remove()* pentru a sterge tot node-ul.

Change the DOM!

Modificarea atributelor, claselor si stilurilor in DOM

Attributes(src, href, class, id, style, data-...)

- Metode folosite:
 - *hasAttribute()* - returneaza boolean
 - *getAttribute()* - returneaza valoarea specifica atributului
 - *setAttribute()* - adauga sau modifica valoarea specifica atributului
 - *removeAttribute()* - sterge un atribut de pe un element.

<http://bit.do/changeAtt>

Change the DOM!

Modificarea atributelor, claselor si stilurilor in DOM

Clase:

- Avem 2 categorii **className** si **classList**
 - *className* - ia sau seteaza valoarea clasei.
 - *classList.add()* - adauga clase
 - *classList.toggle()* - face 'toggle' la o clasa (on off)
 - *classList.contains()* - verifica daca exista o valoare
 - *classList.replace()* - inlocuieste valoarea veche cu una noua.
 - *classList.remove()* - sterge o valoare.
- *ClassList.add()* - adauga o noua clasa pe cand **className** va suprascrie clasa/clasele existente.
- <http://bit.do/changeClass>

Change the DOM!

Modificarea atributelor, claselor si stilurilor in DOM

Styles:

- De obicei stilurile sunt adaugate in fisiere CSS separate dar avem momente cand vrem sa adaugam *inline style* - aici ne ajuta JS
- <http://bit.do/changeStyle>

Obs: Am putea folosi si `setAttribute()` - `div.setAttribute('style', 'border-radius: 50%');` dar ne va sterge tot stilul *inline* deja scris.

PRACTICE:
<http://bit.do/ex1DOM>
<http://bit.do/ex2DOM-bonus>

