HTML, CSS, Javascript

Laborator 23

Limbajul HTML

Resurse:

- https://www.w3schools.com/html/default.asp
- https://developer.mozilla.org/en-US/docs/Web/HTML
- https://html.spec.whatwg.org/multipage/

Limbajul HTML

HTML = Hyper Text Markup Language

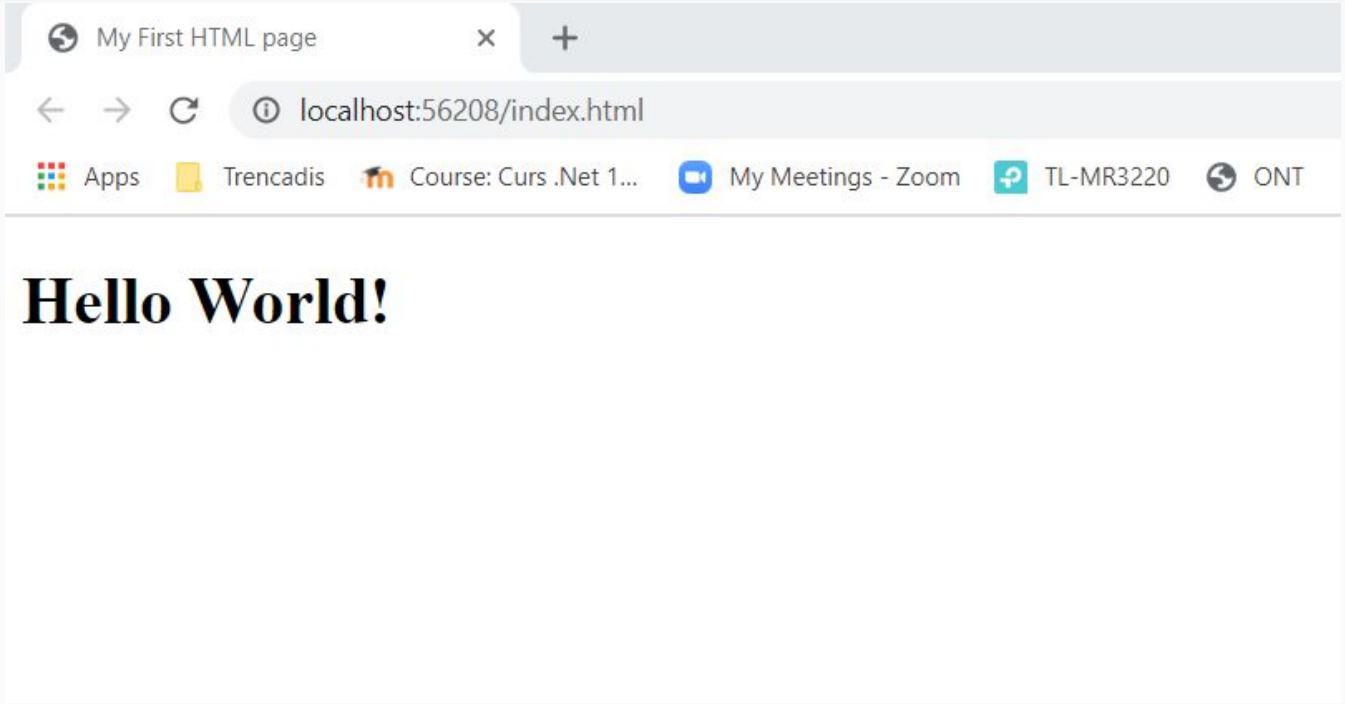
- Este un limbaj declarativ (spunem ce sa faca, nu cum sa faca)
- Utilizat pentru construirea interfetelor grafice care pot fi afisate intr-un browser
- Este un limbaj de markup (bazat pe marcaje): un continut text este structurat folosind un set de reguli de marcare, asa incat sa poata fi procesat in mod automat de catre un software

Limbajul HTML :: Hello World

```
<!DOCTYPE html>

<html lang="en" xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta charset="utf-8" />
    <title>My First HTML page</title>
</head>
<body>
    <h1>Hello World!</h1>
</body>
</html>
```

Limbajul HTML :: Hello World



Limbajul HTML :: Structura unui document HTML

Doctype declaration: prima linie de text din cadrul oricarui document HTML, care (prin continutul sau) specifica versiunea de HTML utilizata:

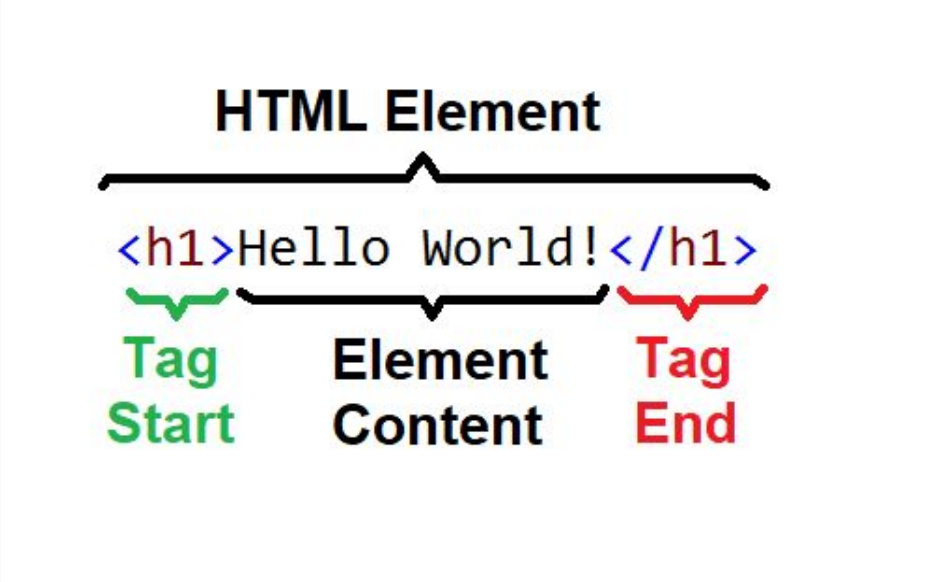
```
<!DOCTYPE html>
```

- Nu face parte efectiv din “limbajul” HTML (vezi DTD) - exista si in alte limbaje de markup (ex: XML) ca o modalitate de a defini cum arata o structura valida a unui document. In XML a fost inlocuit mai recent de catre Xml Schema care ofera mai multe capabilitati de validare
- In HTML a ramas in continuare folosit, browserele bazandu-se pe el pentru a stii ce versiune de HTML urmeaza sa intalneasca, cat de stricta sa fie validarea continutului, etc
- In exemplul nostru, Doctype declaration-ul specifica faptul ca vom utiliza HTML5

<!DOCTYPE html>

Limbajul HTML :: Structura unui document HTML

HTML Elements: un document HTML este alcatuit dintr-o ierarhie de elemente:



<!DOCTYPE html>

Limbajul HTML :: Structura unui document HTML

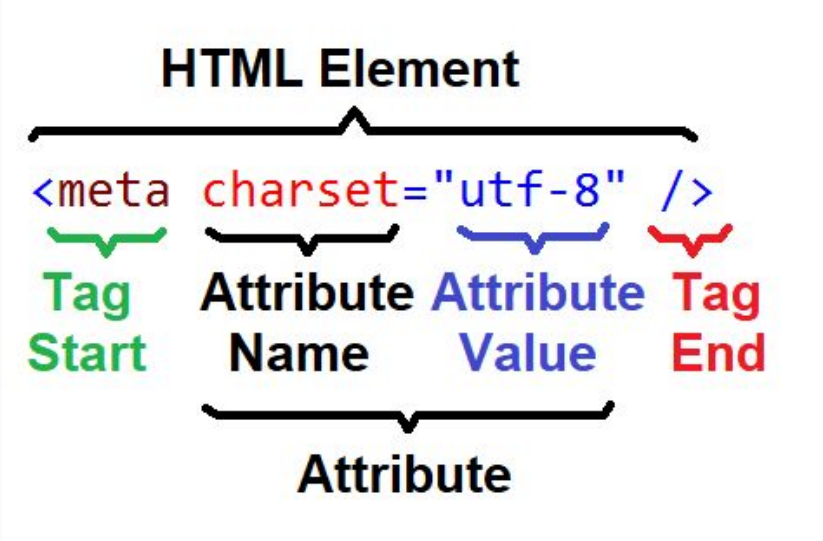
HTML Elements:

- Un element HTML este alcatuit din: un tag de inceput, un continut (optional) si un tag de incheiere
- HTML defineste un set bine precizat de tag-uri
  - Tag-ul <html> este primul tag
  - Tag-urile nu sunt case-sensitive
- Exista elemente HTML care nu au continut. Acestea se inchid folosind o sintaxa de forma:
  - `<tagwithoutcontent />`
- Elementele HTML pot fi imbricate (elemente care sa contina alte elemente)

<!DOCTYPE html>

Limbajul HTML :: Structura unui document HTML

Attributes: sunt o modalitate prin care se poate adauga informatie suplimentara la un element HMTL



<!DOCTYPE html>

Limbajul HTML :: Structura unui document HTML

Attributes:

- Un atribut are: un nume si o valoare
- Valoarea se precizeaza intre ghilimele
- Atributele se definesc intotdeauna pe tag-ul de start
- Atributele pot exista si pe elemente cu continut, si pe elemente fara continut, si pe elemente imbricate
- Exista atribute generale (ex: id, name, class, style, tabindex, etc) care pot fi folosite impreuna cu orice element HTML
- Exista atribute particulare, care au sens doar pentru anumite elemente HTML (ex: href, alt, src, action, etc) - vezi aici o lista completa de atribute

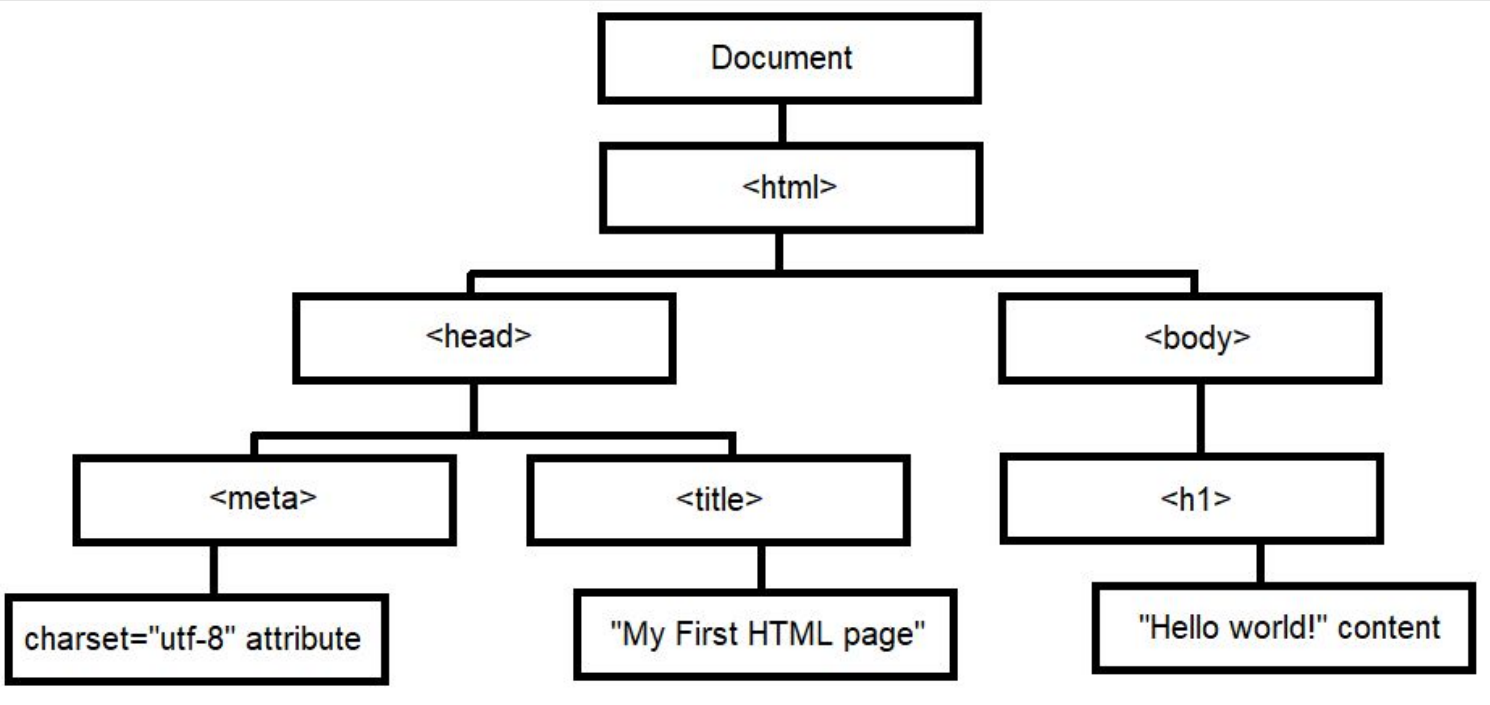
Limbajul HTML :: Structura unui document HTML

DOM: avand in vedere ca elementele HTML pot fi imbricate, ele creeaza o structura ierarhica asemanatoare unui arbore. De fapt, pentru a reprezenta documente HTML, atunci cand incarca o pagina web, browserele chiar asta fac: construiesc o reprezentare ierarhica (arbore) a structurii documentului HTML denumita DOM (Document Object Model) in care:

- Exista o radacina denumita “Document”
- Elementul “html” este primul copil
- Elementele “head” si “body” sunt copii directi ai elementului “html”
- etc

Limbajul HTML :: Structura unui document HTML

DOM:



Limbajul HTML :: Structura unui document HTML

DOM:

- Aceasta structura este expusa apoi in mod programatic de catre browser, putand fi utilizata de alte tehnologii (ex: CSS, JavaScript) pentru stilizare, traversare / cautare elemente, adaugare, editare, stergere elemente, etc

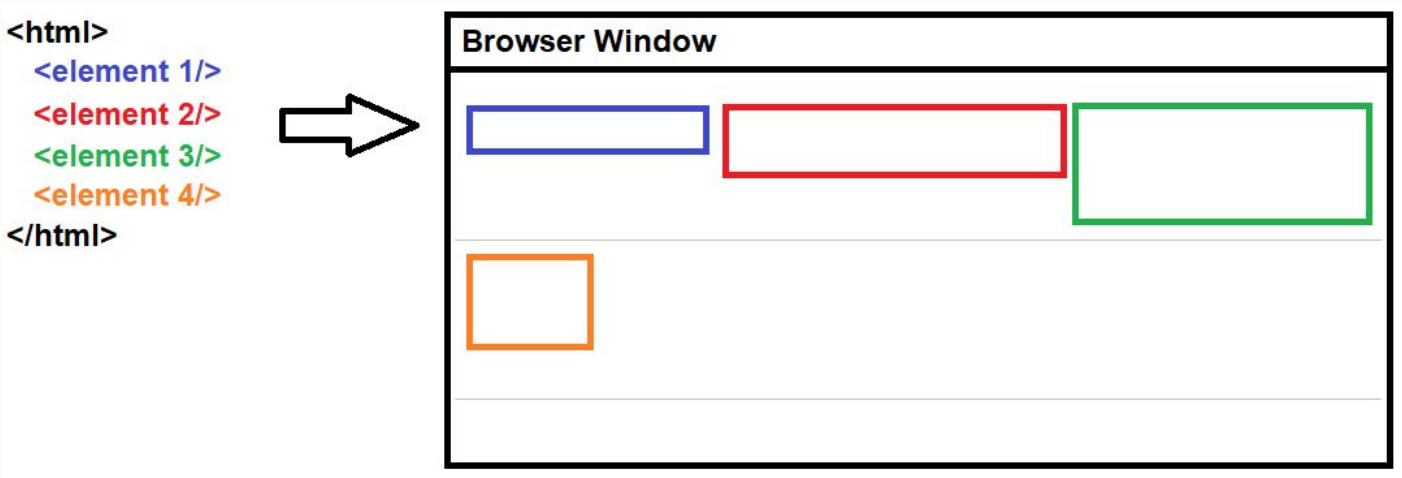
Limbajul HTML :: Reguli de formatare a documentelor

HTML coding rules:

- Intotdeauna incepeti cu declararea DOCTYPE-ului
- Folositi lowercasing (minuscule) pentru elemente / atribute
- Orice tag deschis trebuie si inchis
- Folositi intotdeauna ghilimele pentru valorile atributelor
- Nu puneti spatii intre numele atributului, semnul egal si valoarea atributului
- Evitati liniile prea lungi; Separati blocurile mari de cod cu o linie; Indentati elementele copii ale altor elemente, acolo unde este cazul. Folositi spatii (nu tabs) pentru indentare.
- Nu omiteti elemente importante (html, head, body, title, etc)

Limbajul HTML :: Afisarea elementelor in HTML

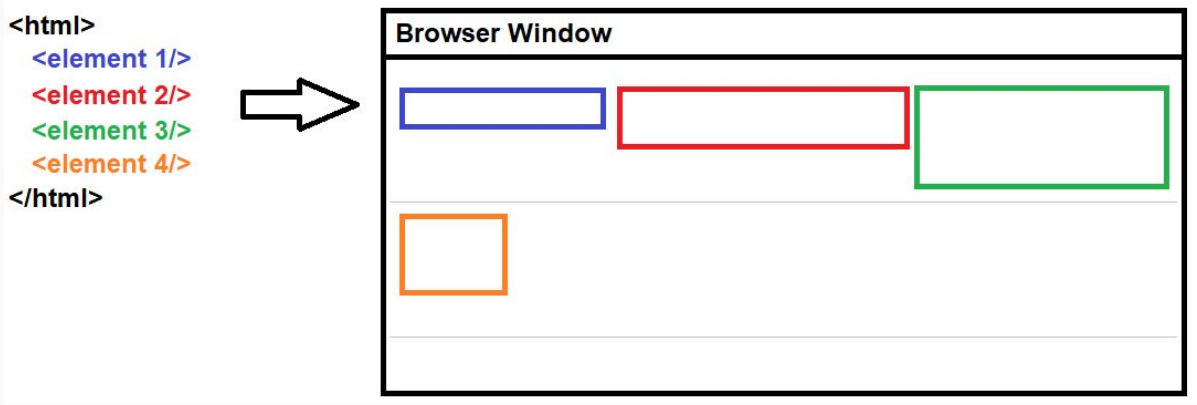
In mod normal browserele randeaza continutul de la stanga la dreapta si de sus in jos, element cu element, in functie de ordinea in care elementele apar in codul sursa:



Limbajul HTML :: Afisarea elementelor in HTML

In functie de tipul elementelor afisate, dupa randarea unui element browserul poate:

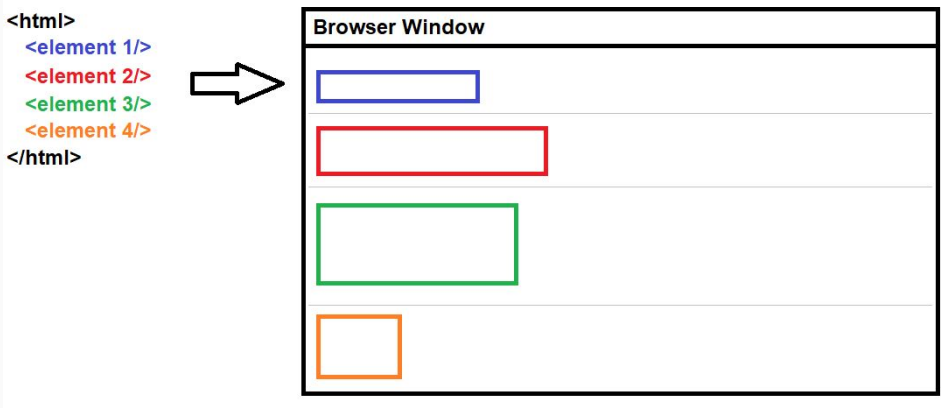
- Continua mai departe, pe aceeasi linie, cu randarea urmatorului element: acest lucru se intampla pentru elementele “**inline**” (a, span, img, select, etc)



Limbajul HTML :: Afisarea elementelor in HTML

In functie de tipul elementelor afisate, dupa randarea unui element browserul poate:

- Va sari pe linie noua si va incepe de la marginea stanga cu randarea urmatorului element, pentru elementele “**block**” (div, form, nav, table, etc)



Limbajul HTML :: Tipuri de elemente :: Elemente utilizate pentru lucrul cu text

Subtitluri (headings): de la <h1> la <h6>, utilizate pentru a structura continutul

Paragrafe: ... , utilizate pentru blocuri mai lungi de text. Atentie la faptul ca HTML elimina spatiile albe / liniile goale multiple.

Linii orizontale: <hr />, utilizate pentru separarea continutului

Linii noi:
, pentru a forta browserul sa randeze continutul care urmeaza incepand de la linie noua

Text pre-formatat: <pre>...</pre>, asemanator paragrafelor, dar pastrand formatarea din codul sursa

Limbajul HTML :: Tipuri de elemente :: Elemente utilizate pentru lucrul cu text

Elemente care modifica formatarea textului: bold, italic, subscript, superscript, important, emphasized text, deleted text, inserted text, etc.

Elemente utilizate pentru citate: citate din alte surse, citat scurt, abrevieri, informatii de adresa, etc.

Caractere care necesita escaping: < (<), > (>),   (spatiu), etc

Simboluri, emoji-uri

Limbajul HTML :: Tipuri de elemente :: Liste

Liste: sunt folosite pentru gruparea unui set de item-uri aflate intr-o anumita relatie.

Exista 2 tipuri principale de liste:

- Liste ne-ordonate:
- Liste ordonate

De asemenea, exista suport pentru liste descriptive: liste care contin un set de termeni, iar pentru fiecare termen se pot oferi una sau mai multe descrieri.

Limbajul HTML :: Tipuri de elemente :: Tabele

Tabele: sunt folosite pentru a aranja informatie sub forma tabelara (randuri / coloane).
Un tabel poate avea:

- Un caption
- Un table header (<thead>)
- Un table body (<tbody>)
- Un table footer (<tfoot>)
- Unul sau mai multe randuri (tr)
- Una sau mai multe coloane (th, sau td)

Limbajul HTML :: Tipuri de elemente :: Continut media

Imagini: folosind tag-ul <img>

Imagini cu zone active de continut (image maps): utilizate de regula pentru un efect de selectie, sau de drill-down pe o imagine.

Imagini de fundal: pentru a seta imagini de fundal pentru anumite elemente HTML.

Imagini provenind din mai multe surse: pentru un plus de flexibilitate in alegerea sursei, in functie de rezolutia dispozitivului utilizat pentru navigare.

Video, audio

Limbajul HTML :: Tipuri de elemente :: Specificarea path-urilor

Tipuri de path-uri:

- Path-uri catre fisiere aflate in acelasi director ca si pagina curenta
- Path-uri catre fisiere aflate intr-un sub-director al directorului in care se gaseste pagina curenta
- Path-uri relative fata de directorul radacina al site-ului
- Path-uri relative fata de directorul curent
- Path-uri absolute

Limbajul HTML :: Tipuri de elemente :: Link-uri catre alte resurse

Hyperlink-uri: permit utilizatorilor sa navigheze de la o pagina la alta.

Link-uri catre resurse externe: utilizate cu precadere pentru incarcarea unor elemente de stil (CSS) folosind fisiere externe.

Limbajul HTML :: Tipuri de elemente :: Elemente de tip “container” pentru alte elemente

Cele mai utilizate elemente cu rol de “container” (elemente care gazduiesc alte elemente) sunt:

- div: un container de tip “block”
- span: un container de tip “inline”, utilizat cu precadere pentru a gazdui text

Limbajul HTML :: Tipuri de elemente :: Containere semantice

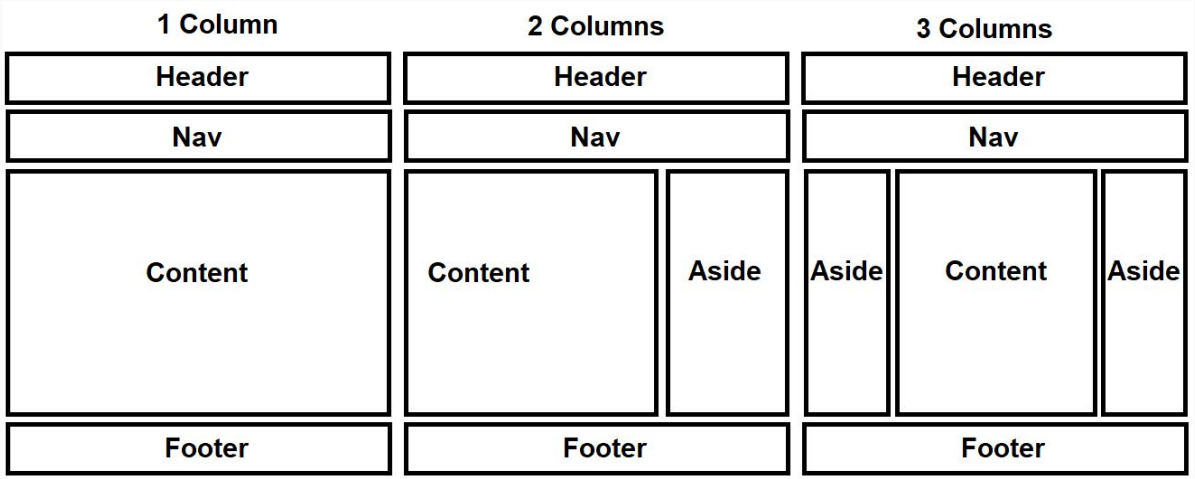
Containerele semantice = acele elemente de tip “container”, al caror nume denota si o anumita semnificate a continutului.

Exemple:

- Article: defineste o zona de continut independent, intr-un document / pagina / aplicatie / site, continut care poate fi inclusiv distribuit / refolosit, e.g.: un blog post
- Section: defineste o sectiune intr-un document
- Header: o zona de antet, definita de regula ca si primul “copil” al unei zone de continut (ex: article)
- Footer: similar cu header, dar cu intentia de a reprezenta zona de subsol
- Nav: reprezinta o zona care contine elemente ce tin de navigatia in cadrul documentului / paginii / site-ului (in principal link-uri)
- Aside: reprezinta o zona de continut adiacenta continutului principal
- Etc ...

Limbajul HTML :: Construirea de layout-uri in HTML, Responsive design

Elementele de tip container (semantice, sau generice) sunt folosite de regula pentru construirea de layout-uri (structura paginii / site-ului). Exemple de layout-uri uzuale:



Limbajul HTML :: Construirea de layout-uri in HTML, Responsive design

Pentru realizarea acestor layout-uri este nevoie sa se foloseasca HTML + CSS si de regula se apeleaza la una din urmatoarele tehnici:

- HTML + folosirea unui framework CSS (ex: Bootstrap)
- HTML + CSS custom folosind proprietatea “float” (defineste directia de “plutire” a elementelor, in cadrul flow-ului paginii)
- HTML + CSS custom folosind CSS3 flexbox
- HTML + CSS custom folosind CSS3 grid layout

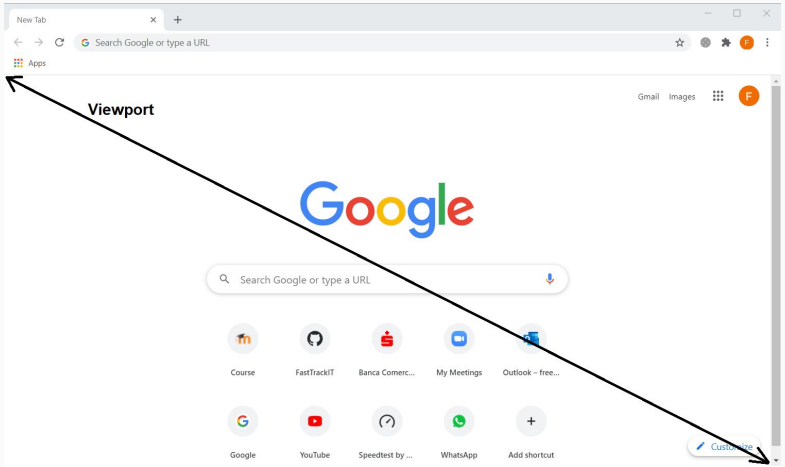
Limbajul HTML :: Construirea de layout-uri in HTML, Responsive design

Responsive design = un design / layout HTML care se ajusteaza automat in functie de dimensiunea device-ului de pe care se face browsing, asa incat interfata rezultata sa arate bine / sa fie utilizabila pe orice fel de device.

Responsive design se implementeaza folosind HTML + CSS media queries, care permit specificarea de stiluri diferite in functie de dimensiunea ecranului pe care se face browsing-ul.

Limbajul HTML :: Construirea de layout-uri in HTML, Responsive design

Viewport = suprafata vizibila a paginii web, asa cum este ea perceputa de utilizator pe ecranul pe care face browsing. Viewport-ul poate diferi de la un device la altul si va fi mai mic de exemplu pe un telefon mobil decat pe un ecran de desktop:



Limbajul HTML :: Construirea de layout-uri in HTML, Responsive design

Pentru un design responsive este important sa se seteze dimensiunea viewport-ului asa incat sa se potriveasca cu rezolutia device-ului pe care este afisata pagina HTML, in absenta acestui pas, browserul va presupune niste dimensiuni predefinite ale viewport-ului (care sunt mai potrivite pentru ecrane desktop, decat pentru device-uri mobile), lucru care va duce la necesitatea de a efectua suplimentar si o scalare a interfetei grafice, rezultatul aratand cam asa (see next slide):


Lorem Ipsum Dolor
Lorem Ipsum Dolor
Search...
Lorem Ipsum Dolor
Dolor sit amet, consectetur adipiscing elit. Aenean non erat sit amet sem fringilla vehicula vel ut ex.

Limbajul HTML :: Construirea de layout-uri in HTML, Responsive design

Pentru a seta view-port-ul automat la o dimensiune potrivita cu rezolutia ecranului device-ului de pe care se face browsing se va folosi urmatorul cod HTML in zona de antet (head) a documentului:

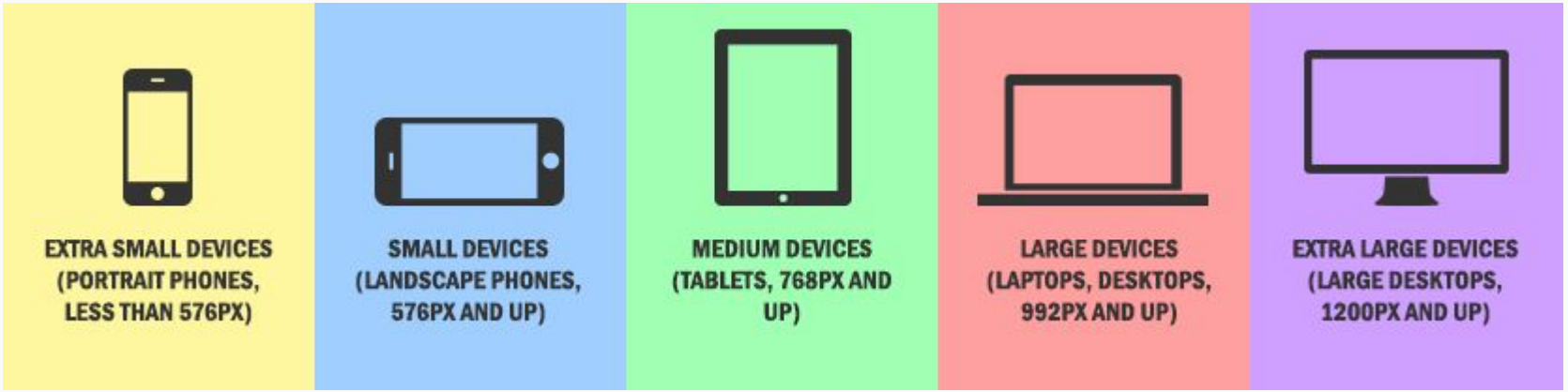
```
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
```

Apoi, un design responsive va respecta urmatoarele principii:

- Nu se vor folosi elemente cu dimensiuni fixe mari (risca sa apara scroll orizontal)
- Nu se va presupune o anumita dimensiune a view-port-ului pentru ca interfata sa arate asa cum trebuie
- Se vor utiliza CSS media queries pentru a ajusta interfata in functie de dimensiunea ecranului

Limbajul HTML :: Construirea de layout-uri in HTML, Responsive design

Rezolutii uzuale (ex: Bootstrap):



Limbajul HTML :: Formulare HTML

Pe langa elementele prezentate pana acum, care au mai mult rol de a defini layout-ul si de a prezenta informatie pe o pagina, HTML permite si construirea de formulare, cu ajutorul carora se pot colecta date introduse de catre utilizator.

Formularele se realizeaza folosind tag-ul <form>. Caracteristici importante ale unui form:

- Atributul “action”: specifica URL-ul catre care se vor trimite (submit) datele colectate prin formular
- Atributul “method”: specifica metoda HTTP (GET sau POST) prin care se vor trimite datele

Limbajul HTML :: Formulare HTML

Formularele se realizeaza folosind tag-ul <form>. Caracteristici importante ale unui form:

- Elementele de tip: <input>, <textarea>, <button>, <select> definite in cadrul <form>-ului vor colecta datele de la utilizator.
  - Pentru ca datele sa fie culese & trimise, aceste elemente trebuie sa aibe atributul “name”
  - In functie de metoda HTTP aleasa browserul va trimite datele astfel:
    - Pentru GET: datele vor fi atasate in formatul query-string la url-ul specificat in action (ex: https://somesite.com?name1=value1&name2=value2)
    - Pentru POST: datele vor fi incluse in body-ul request-ului HTTP, in mod uzual intr-un format denumit “x-www-form-urlencoded” (name1=value1&name2=value2), sau pentru cazul in care este nevoie si de upload de fisiere in formatul “multipart/form-data” - vezi aici o analiza a celor 2 formate
- Alte elemente cu rol de “etichetare” a elementelor de mai sus: <label>, <fieldset>
- Alte elemente HTML cu rol de formatare a continutului / layout-ului (divs, spans, etc)

Limbajul HTML :: Formulare HTML

Cel mai uzual element intalnit in cadrul formularelor este elementul <input>. Caracteristici importante:

- Atributul “name”: specifica numele elementului, care va fi folosit ca si o cheie atunci cand se vor “impacheta” si trimite datele catre URL-ul specificat in “action”-ul form-ului
- Atributul “id”: un identificator al elementului, folosit in special atunci cand elementul <input> are si un label text cu atribut “for” (“for” va trebui sa specifice identificatorul elementului caruia i se aplica eticheta text)
- Atributul “type” care specifica tipul elementului. HTML suporta urmatoarele tipuri de input-uri:
  - “text”: reprezinta un textbox. Este valoarea implicita pentru “type”
  - “button”: reprezinta un buton. Nu influenteaza datele trimise, este folosit de regula pentru a declansa executia unui cod JavaScript
  - “checkbox”: un checkbox

Limbajul HTML :: Formulare HTML

Cel mai uzual element intalnit in cadrul formularelor este elementul <input>. Caracteristici importante:

- Atributul “type” care specifica tipul elementului. HTML suporta urmatoarele tipuri de input-uri:
  - “color”: un color picker
  - “date”: un date-picker
  - “datetime-local”: un date & time picker
  - “email”: un camp pentru introducerea unei adrese email (se va face automat si validare pe submit)
  - “file”: un control pentru upload de fisiere
  - “hidden”: un camp text care nu este afisat (utilizat doar ca un fel de mecanism de stocare a unor informatii, fara a oferi utilizatorului posibilitatea de a-l edita. Totusi, poate fi modificat din JavaScript / browser development tools!)

Limbajul HTML :: Formulare HTML

Cel mai uzual element intalnit in cadrul formularelor este elementul <input>. Caracteristici importante:

- Atributul “type” care specifica tipul elementului. HTML suporta urmatoarele tipuri de input-uri:
  - “image”: permite submit-ul formularului, afisand un control de tip “buton-imagine”. Nu influenteaza datele trimise
  - “month”: permite introducerea / selectia anului si a lunii
  - “number”: permite introducerea de valori numerice
  - “password”: permite introducerea de parole. In mod implicit caracterele sunt mascate, insa continutul poate fi citit in clar din JavaScript / browser development tools
  - “radio”: defineste un radio button. Radio button-urile permit o singura selectie dintr-un grup de mai multe optiuni. Pentru a grupa mai multe radio-button-uri se va folosi acelasi “name”
  - “range”: defineste un slider care permite selectia unui interval de valori

Limbajul HTML :: Formulare HTML

Cel mai uzual element intalnit in cadrul formularelor este elementul <input>. Caracteristici importante:

- Atributul “type” care specifica tipul elementului. HTML suporta urmatoarele tipuri de input-uri:
  - “reset”: un buton care permite resetarea continutului formularului la valorile initiale
  - “search”: un textbox cu rol de camp de cautare
  - “submit”: un buton care permite submit-ul formularului
  - “tel”: un camp care permite introducerea unui numar de telefon (se va face automat validare de pattern la submit)
  - “time”: un camp care permite introducerea / selectia unei ore & minut
  - “url”: un camp care permite introducerea unui URL (se va face automat validare la submit)
  - “week”: permite selectia unei saptamani

Limbajul HTML :: Exercitiu

Exercitiu: proiectati un formular de register pentru un site de recrutare, formularul urmand sa contina

- Adresa email
- Nume si prenume
- Data Nasterii
- Sexul (folosind radio buttons, sau select)
- Numarul de telefon
- Link-ul catre profilul de Facebook / LinkedIn

CSS

Resurse:

- https://www.w3schools.com/css/default.asp
- https://developer.mozilla.org/en-US/docs/Web/CSS/Tutorials

CSS

CSS = Cascading Style Sheets

- Este un limbaj care descrie stilurile aplicate unui document / pagini
- Daca rolul HTML-ului este sa structureze informatia dintr-un document, rolul CSS-ului este sa o aranajeze / formateze grafic asa incat sa arate bine
- La fel ca si HTML, este un limbaj declarativ (spunem cum dorim sa arate un element, nu ce anume este de facut pentru ca acel element sa arate asa)
- A fost proiectat ca o adaugare la HTML, cu scopul de a obtine o separare mai buna a continutului de prezentare, pentru a avea astfel mai multa flexibilitate pe partea de prezentare (acelasi continut poate fi prezentat in mai multe moduri)

CSS

Structura generala a unui document CSS:

- Un document CSS contine de regula 1 sau mai multi selectori, selectorii avand rolul de a identifica elementul / elementele carora urmeaza sa li se aplice formatarea de stil specificata
- Pentru fiecare selector, proprietatile de stil si valorile lor

```
selector {
    style-property1: value1;
    style-property2: value2;
    ...
}
```

CSS :: Selectori

Selector = descrie o regula de identificare a unuia sau mai multor elemente HTML pentru care urmeaza sa se aplice setul de formatari specificate.

Tipuri de selectori:

- Selectori simpli
- Selectori combinati
- Selectori folosind pseudo-clase
- Selectori folosind pseudo-elemente
- Selectori folosind atribute

CSS :: Selectori :: Selectori simpli

Selectori simpli

- Selectorul universal (*): specifica o regula de stil care se aplica tuturor elementelor HTML din pagina
- Selectorul de tag HTML (numele tag-ului HTML): specifica o regula de stil aplicabila tuturor elementelor HTML avand tag name-ul specificat
- Selectorul de id HTML (#id): specifica o regula de stil aplicabila elementului avand identificatorul (atributul id) specificat
- Selectorul de clasa (.class): specifica o regula de stil aplicabila elementelor avand clasa (atributul class) specificata

CSS :: Selectori :: Selectori combinati

Selectori combinati

- Selectorul parinte - descendent (parent descendant): specifica o regula de stil care se aplica tuturor elementelor care sunt descendenti (identificati prin selectorul “descendant”) ai elementelor parinte (identificati prin selectorul “parent”)
- Selectorul parinte - copil imediat: (parent > child): la fel ca pentru parinte-descendenti, insa de data aceasta elementele “copii” trebuie sa fie descendent direct (copil) al elementului parinte - vezi aici un exemplu de parent - child vs descendant
- Selectorul “frate adiacent” (element + sibling): specifica o regula care se aplica tuturor elementelor care sunt “frati adiacenti” (identificate prin selectorul “sibling”) ai elementelor identificate prin selectorul “element”. “Frate adiacent” inseamna:
  - Frate = au acelasi element parinte
  - Adiacent = urmeaza imediat dupa elementul specificat

CSS :: Selectori :: Selectori combinati

Selectori combinati

- Selectorul “frate general” (element ~ sibling): specifica o regula care se aplica tuturor elementelor care sunt “frati” (identificate prin selectorul “sibling”) in sensul general (au acelasi parinte) ai elementelor identificate prin selectorul “element”. Vezi aici un exemplu.

CSS :: Selectori :: Selectori folosind pseudo-clase

Selectori folosind pseudo-clase

O pseudo-clasa este un identificator special care defineste starea unui element HTML (ex: link vizitat / ne-vizitat, element peste care trece cursorul mouse-ului, elment care are focus, element afisat intr-o anumita limba, element disabled, etc).

Sintaxa generala a unui asfel de selector este:

```
selector:pseudo-class {
    style-property1: value1;
    style-property2: value2;
    ...
}
```

CSS :: Selectori :: Selectori folosind pseudo-clase

Selectori folosind pseudo-clase

Exemple uzuale:

- Link nevizitat: a:link
- Link vizitat: a:visited
- Div peste care se “plimba” mouseu-ul: div:hover
- Vezi aici o lista cu mai multe exemple

CSS :: Selectori :: Selectori folosind pseudo-elemente

Selectori folosind pseudo-elemente

Un pseudo-element este un identificator special care defineste fie o parte a unui element HTML existent (prima litera, prima linie, portiunea selectata), fie un element virtual (continut inainte / dupa un element HTML existent).

Sintaxa generala a unui asfel de selector este:

```
selector::pseudo-element {
    style-property1: value1;
    style-property2: value2;
    ...
}
```

CSS :: Selectori :: Selectori folosind pseudo-elemente

Selectori folosind pseudo-elemente

Exemple uzuale:

- Prima litera din textul unui paragraf: p::first-letter
- Continut inaintea unui div: div::before
- Continut dupa un div: div::after
- Vezi aici o lista cu mai multe exemple

CSS :: Selectori :: Selectori folosind attribute

Selectori folosind atribute

- Selector elemente avand un atribut a carui valoare este egala cu o valoare specificata (element[attribute=”value”])
- Selector elemente avand un atribut a carui valoare contine o valoare specificata (element[attribute~=”value”])
- Selector elemente avand un atribut a carui valoare incepe cu o valoare specificata (element[attribute|=”value”]). In acest caz match-ul se realizeaza doar pentru cazul in care valoarea atributului este un cuvant intreg, ex: div[class|=”top”] match-uieste div-ul care are class=”top”, sau class=”top-ten”, dar nu si div-ul care are class=”top100”. Vezi aici un exemplu.
- Selector elemente avand un atribut a carui valoare incepe cu o valoare specificata (element[attribute^=”value”]) - se foloseste atunci cand dorim matching si in cazul in care valoarea atributului nu este un cuvant intreg (string.StartsWith clasic)
- Etc, vezi mai multe exemple aici.

CSS :: Mostenirea in CSS

Mostenirea in CSS: in CSS, conceptul de “mostenire” se refera la faptul ca elementele “copil” preiau stilurile care se aplica la nivelul elementelor “parinte” (termenul de “mostenire” de-aici este de fapt legat de eticheta “cascading” din numele CSS: in absenta unei reguli explicite care sa schimbe o anumita proprietate CSS, in general proprietatile definite in “amonte”/upstream se “revarsa” ca intr-o cascada si catre elementele din “aval”/downstream).

Proprietati ne-mostenite: anumite proprietati CSS nu se mostenesc in mod implicit, ci trebuie specificate in mod explicit si pentru descendenti - ex: “border-style”. O lista completa cu proprietatile mostenite - vs - nemostenite o gasiti in standardul CSS.

CSS :: Mostenirea in CSS

Valori initiale: standardul CSS specifica o valoare initiala pentru fiecare proprietate CSS. Atunci cand se doreste “resetarea” unei proprietati la acea valoare initiala se poate folosi cuvantul cheie “initial”.

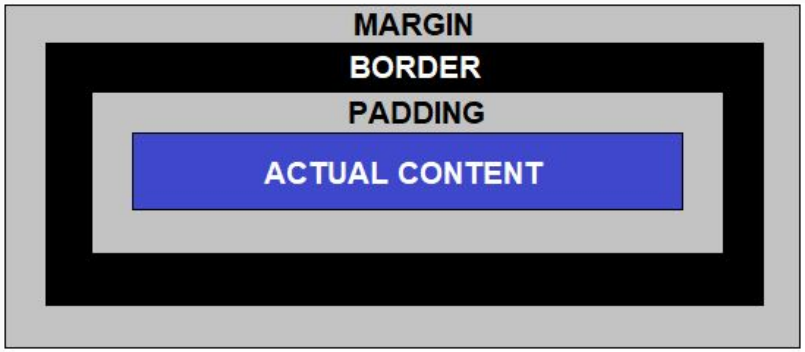
Observatie 1: pentru proprietatile mostenite resetarea are loc doar la nivelul elementului “radacina”, elementele descendenti urmand sa foloseasca mostenirea mai departe.

Observatie 2: a nu se confunda valoarea initiala din standardul CSS, cu valoarea initiala pe care o seteaza browserul. Exista asa numitele “user agent stylesheets”, prin care fiecare browser isi poate defini un set de customizari proprii, pentru valorile initiale, customizari care se aplica implicit in absenta altor specificatii.

CSS :: CSS Box Model

CSS Box Model: din punct de vedere al randarii grafice, orice element HTML poate fi considerat “o cutie” care are urmatoarele proprietati:

- O margine (distanta fata de alte elemente)
- Un chenar (o linie care incadreaza continutul elementului)
- O zona tampon / padding (o distanta intre chenar si continutul propriu zis)
- Continutul efectiv



CSS :: CSS Box Model

Intelegerea acestui model de randare este importanta atunci cand se lucreaza cu proprietatile CSS de latime (width) si inaltime (height), intrucat acestea opereaza doar asupra zonei de content, insa pentru a calcula cat loc ocupa efectiv un element HTML pe layout, trebuie avute in vedere si celelalte proprietati (padding, borders, margins).

Observatie: exista si un model alternativ, in care proprietatile de width si height opereaza asupra content + padding + borders, insa el nu este folosit in mod implicit, ci pentru a-l utiliza, programatorul trebuie sa specifice in mod explicit faptul ca doreste ca proprietatea “box-sizing” sa fie “border-box”. In mod traditional IE folosea acest model, lucru care ducea la inconsistente de rendering.

CSS :: CSS Positioning

CSS defineste proprietatea “position” care poate specifica unul din urmatoarele moduri de pozitionare a elementelor:

- static: valoarea implicita, pozitionare normala, in conformitate cu flow-ul elementelor definite in pagina
- relative: pozitionare relativa fata de pozitia normala pe care ar ocupa-o in flow-ul paginii
- fixed: pozitionare relativa la viewport-ul paginii. Elementele astfel pozitionate raman in acelasi loc, chiar daca userul scroleaza in pagina
- absolute: pozitionare absoluta fata de elementul parinte / stramos care are specificata o pozitionare (altceva decat “static”).

CSS :: CSS in HTML

Stilurile CSS pot fi aplicate elementelor HTML folosind unul din urmatoarele moduri:

- Folosind atributul general style (atribut aplicabil oricarui element HTML). Acest gen de a include informatie de stil impreuna cu restul atributelor HTML poarta denumirea de stil “inline”. In general este **nerecomandat** pentru ca:
  - Amesteca continutul cu prezentarea
  - Este foarte greu de customizat
- Folosind tag-ul <style> din HTML. CSS-ul va fi in continuare continut in cadrul documentului HTML, dar cel putin exista o separare mai clara a stilurilor de restul continutului.
- Folosind tag-ul <link> si incarcand informatia de stil dintr-ul fisier css separat. Aceasta varianta ofera cea mai clara separare intre continut si prezentare. Browserul poate face caching la fisierele CSS externe, lucru care poate fi un avantaj important (dar uneori si o problema!)

CSS :: CSS specificity

Exista posibilitatea ca atunci cand lucram cu CSS (ex: din cauza folosirii unor librarii CSS pentru care avem nevoie sa facem customizari, sau pentru implementarea unui suport pentru teme multiple) sa avem nevoie sa folosim reguli CSS care suprascriu alte reguli CSS, caz in care ne va interesa sa stim cum anume rezolva browserele cazul in care exista potential mai multe reguli in conflict pentru acelasi element HTML.

CSS :: CSS specificity

Pentru rezolvarea conflictelor intre reguli, standardul CSS defineste o ierarhie a prioritatilor (top to bottom):

1. O regula care se termina in “!important” suprascrie orice alta regula
2. Regulile definite in stilul inline (atributul style din din HTML)
3. Regulile definite pentru un ID de element HTML
4. Regulile definite pentru clase, pseudo-clase, sau atribute
5. Regulile definite pentru elemente si pseudo-elemente

Alte observatii:

- Pentru regulile compuse, prioritatile se compun. Astfel o regula mai “lunga” (a se citi mai specifica) va “castiga” in fata uneia mai simple / generice (ex: clasa, sau element)
- De asemenea, pentru reguli avand o prioritate egala, castiga ultima regula intalnita in codul sursa.

CSS :: CSS specificity

Calculator de specificitate:

Regula	4	3	2	1	0
	!important	inline CSS	#ID	.class :pseudo-class [attr=”val”]	element ::pseudo-element
Valoare	10,000	1,000	100	10	0

CSS :: CSS specificity

Reguli de buna practica:

- Atentie la utilizarea “!important” - de evitat la stiluri generale / site wide / themes / plugins
- Atunci cand dezvoltati plugin-uri si/sau theme care pot fi customizate, lasati posibilitatea ca utilizatorul sa poata aduce modificari prin cresterea specificitatii (ex: folositi reguli pentru elemente, folosind atribute, sau clase, evitati reguli cu specificitate mare cum sunt cele dupa ID)
- Daca folositi librarii CSS, asigurati-va ca stilurile custom se incarca in pagina dupa stilurile din librariile CSS, asa incat sa puteti face customizari fara “!important”
- Folositi-va de regulile de specificitate pentru a face customizari la stilurile mostenite din alte librarii

CSS :: CSS naming conventions

“*There are only two hard things in Computer Science: cache invalidation and naming things.*” - Phil Karlton

Problema:

- Cu cat avem mai multe reguli CSS, cu atat devine mai greu de urmarit ce stil, carui element se aplica, in ce conditii, etc
- A alege nume de clase care sa fie suficient de clare ca sa fie usor de inteles ce anume fac, dar in acelasi timp suficient de flexibile pentru a putea fi refolosite in mai multe situatii este un task dificil

CSS :: CSS naming conventions :: BEM

BEM este o suita de reguli de denumire a selectorilor CSS bazata pe principiile urmatoare:

- Entitati:
  - **B**lock: o entitate de sine statatoare
  - **E**lement: o parte a unui block
  - **M**odificator: un flag aplicat unui block, sau unui element, care semnaleaza o anumita schimbare
- Regula de denumire a unei clase:
  - “block”
  - “block--modifier”
  - “block__element”
  - “block__element--modifier”
- Un tutorial dragut despre BEM aici

CSS :: CSS frameworks

CSS Frameworks: pentru consistenta in aplicarea stilurilor in diferite browsere (care au fiecare particularitatile lor si modul lor propriu de a interpreta anumite directive) cat si pentru a usura anumite task-uri au fost creeate framework-uri CSS.

De regula ce avem de facut pentru a folosi un framework CSS este sa includem in pagina web unul sau mai multe fisiere CSS (posibil si JavaScript!) folosind tag-uri de <link> si <script>, sau daca folosim tool-uri de development client-side (Node-JS + npm) sa instalam pachetele care contin fisierele cu codul CSS / javascript aferent framework-ului respectiv.

CSS :: CSS frameworks

Exemple de framework-uri CSS:

- Bootstrap: ajuta foarte mult pentru creearea de design-uri responsive, de asemenea ofera o paleta mare teme, iconite, componente grafice (butoane stilizate, carousel, meniuri, etc)
- Materialize (material design CSS framework)
- Foundation

CSS :: Holy Grail Layout

Exercitiu: rezolvati problema clasica a layout-ului cu 3 coloane (Holy Grail Layout)
Va puteti inspira din urmatoarele rezolvari:

- Flexbox
  - https://philipwalton.github.io/solved-by-flexbox/demos/holy-grail/
  - https://www.developerdrive.com/holy-grail-layout-flexbox/
- CSS grid
  - https://www.digitalocean.com/community/tutorials/css-css-grid-holy-grail-layout

Javascript

JavaScript: un limbaj de programare, conceput initial ca un limbaj de tip “script”, care era interpretat de browser si care ajuta la obtinerea de interactiuni mai complexe cu DOM-ul paginii HTML, insa care in timp a devenit un limbaj de programare standardizat (ECMAScript-262), generalist, putand fi folosit atat pentru dezvoltarea de functionalitati in cadrul paginilor web (front-end development) cat si pentru dezvoltarea de alte aplicatii (desktop, mobile, servicii back-end, etc).

Observatie: Fiind un subiect atat de vast (un alt limbaj de programare!), pentru cursul nostru ne va interesa sa stim cum putem folosi suportul JavaScript oferit de browsere, pentru imbunatatirea / realizarea aplicatiilor web mai complexe.

Javascript

Caracteristici: asemanari si deosebiri fata de C#

	Asemanari	Desebiri
Generalitati	• Sintaxa oarecum familiara, data de faptul ca o inspiratie comuna (C/C++, Java) • Suporta paradigma OOP	• Javascript este un limbaj interpretat de browser • In Javascript OOP-ul este bazat pe prototipuri (abia mai recent exista suport si pentru clase)
Type System	• Exista un set de tipuri de date primitive (desi acesta este mai limitat in JavaScript fata de C#)	• Javascript este un limbaj dinamic (tipurile variabilelor sunt date de continutul acestora, ele se pot inclusiv schimba pe durata executiei) • Orice expresie are ca rezultat un obiect Ex: o functie este un obiect care are metode “apply” si “call” • Obiectele se comporta ca niste dictionare, proprietatile putand fi accesate dupa nume, sau putand fi iterate

Javascript

Caracteristici: asemanari si deosebiri fata de C#

	JavaScript	C#
Utilizat pentru	• In special pentru web development, impreuna cu HTML si CSS • A raspunde la evenimente, ca urmare a interactiunii utilizatorului cu pagina web • A manipula (cautare, adaugare, modificare, stergere) DOM-ul paginii HTML • A apela alte pagini / servicii web folosind protocolul HTTP, sincron sau asincron (Ajax) • A lucra cu diverse servicii oferite de browser (canvas, local storage, console, cache, geolocation, notifications, webRTC, etc)	• Pentru a dezvolta diverse aplicatii, fara a fi limitat la aplicatii web • In general utilizat pentru dezvoltarea partii de “server” a aplicatiilor web (obs: odata cu Blazor, C# permite si dezvoltare front-end, insa tehnologia este la inceput, ramane sa vedem cat succes va avea in a inlocui JavaScript 😁) • Access servicii de baza oferite de sistemul de operare (mediate de runtime-ul CLR), cum ar fi: file system, sockets and networking, threads, processes, etc

Javascript :: Hello World!

```
<!DOCTYPE html>

<html lang="en" xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta charset="utf-8" />
    <title>Title</title>
    <script type="text/javascript">
        function onPageLoaded() {
            alert("Hello World!");
        }

        window.onload = onPageLoaded;
    </script>
</head>
<body>
    <p>My HTML content ...</p>
</body>
</html>
```

Javascript - tipuri de date

Tipuri primitive de date in JavaScript

- Undefined
- Boolean
- Number
- String
- BigInt
- Symbol
- Null
- Object
- Function
- Array

Javascript - operatori

Operatori in JavaScript

- Assignmnent: =
- Aritmetici: +, -, *, /, %, ++ (increment), -- (decrement)
- Compound assignment: +=, -=, *=, /=, %=
- String concatenation: +, +=
- Bitwise: &, |, ~ (not), ^ (xor), <<, >>
- Logical: &&, ||, !
- Conditional: ==, ===, !=, !==, <, >, <=, >=
- Type operators: typeof, instanceof

Javascript - blocuri de control al executiei

[Instructiuni pentru controlul executiei in JavaScript]

- Conditional statements:
  - if (condition) { ... } else { ... }
  - switch (expression) { case label1: [statements] break; ... }
- Loops
  - for (initializer;condition;iterator) { ... }
  - while (condition) { ... }
  - do { ... } while(condition)
  - break & continue
- Error handling
  - try { ... } catch(err) { ... } finally { ... }
- Return

Javascript - Variabile

Lucrul cu variabile in JavaScript

- Global Scope, sau function scope: variabilele globale, sau in function scope (dupa caz) se declara cu “var”
  - Obs: JavaScript face variable hoisting pentru variabilele declarate cu var!
- Local variable, sau block scope: se declara cu “let”
- Immutable local variable, sau block scope: se declara cu “const”

Javascript - Functii

Functii in JavaScript

- O functie este un bloc de cod care executa un anumit task si care poate fi apelat folosind numele (si eventual o lista de parametrii)
- In JavaScript functiile se declara cu function:

```
function sum(a, b) {
    return a + b;
}
```

- Functiile se invoca folosind paranteze

```
var result = sum(2, 3);
```

- In JavaScript functiile sunt obiecte: au un prototip, au proprietati, au metode, etc

Javascript - Obiecte

Lucrul cu obiecte in JavaScript

Exista mai multe stiluri pentru a defini obiecte in JavaScript:

- Obiectele pot fi create ex-nihilo

```
var person = new Object();
person.FirstName = "John";
person.LastName = "Doe";
person.sayHello = function () {
    return "My name is " + this.FirstName + " " + this.LastName;
}

function onPageLoaded() {
    console.log(person);
    console.log(person.sayHello());
}

window.onload = onPageLoaded;
```

```
var person = {};
person.FirstName = "John";
person.LastName = "Doe";
person.sayHello = function () {
    return "My name is " + this.FirstName + " " + this.LastName;
}

function onPageLoaded() {
    console.log(person);
    console.log(person.sayHello());
}

window.onload = onPageLoaded;
```

Javascript - Obiecte

Lucrul cu obiecte in JavaScript

Exista mai multe stiluri pentru a defini obiecte in JavaScript:

- Obiectele pot fi create folosind o sintaxa asemanatoare dictionarelor

```
var person = {};
person["FirstName"] = "John";
person["LastName"] = "Doe";
person["sayHello"] = function () {
    return "My name is " + this.FirstName + " " + this.LastName;
}

function onPageLoaded() {
    console.log(person);
    console.log(person.sayHello());
}

window.onload = onPageLoaded;
```

Javascript - Obiecte

Lucrul cu obiecte in JavaScript

Exista mai multe stiluri pentru a defini obiecte in JavaScript:

- Obiectele pot fi create folosind notatia JSON

```
var person = {
    "FirstName": "John",
    "LastName": "Doe",
    "sayHello": function () {
        return "My name is " + this.FirstName + " " + this.LastName;
    }
};

function onPageLoaded() {
    console.log(person);
    console.log(person.sayHello());
}

window.onload = onPageLoaded;
```

Javascript - Obiecte

Lucrul cu obiecte in JavaScript

Exista mai multe stiluri pentru a defini obiecte in JavaScript:

- Obiectele pot fi create folosind functii (similar claselor cu constructor din C#)

```
var Person = function (firstName, lastName) {
    this.FirstName = firstName || "";
    this.LastName = lastName || "";
    this.sayHello = function () {
        return "My name is " + this.FirstName + " " + this.LastName;
    }
}

function onPageLoaded() {
    var person = new Person("John", "Doe");
    console.log(person);
    console.log(person.sayHello());
}

window.onload = onPageLoaded;
```

Javascript - Obiecte

Lucrul cu obiecte in JavaScript

Exista mai multe stiluri pentru a defini obiecte in JavaScript:

- Obiectele pot fi create folosind clase (relativ nou in JavaScript)

```
class Person {
    constructor(firstName, lastName) {
        this.FirstName = firstName || "";
        this.LastName = lastName || "";
    }

    sayHello() {
        return "My name is " + this.FirstName + " " + this.LastName;
    }
};

function onPageLoaded() {
    var person = new Person("John", "Doe");
    console.log(person);
    console.log(person.sayHello());
}

window.onload = onPageLoaded;
```

Javascript - OOP

OOP in JavaScript

- Pana nu de mult, in JavaScript nu exista conceptul de clasa. OOP-ul era implementat folosind prototipuri (obiect parinte pentru obiectul curent) si functii
  - Mostenirea era una de prototip: pentru obiectele creeate de o functie, se putea seta un prototip (obiect) comun
  - Functiile virtuale erau simulate prin apelul metodelor din prototip si/sau din implementarea curenta
- Incepand din 2015 in standardul ECMAScript a fost adaugat si suport pentru clase, insa aceastea functioneaza mai mult ca un syntactic sugar peste implementarea traditionala de OOP din JavaScript (prototype based)

Javascript - JavaScript si HTML

Putem implementa cod JavaScript in cadrul paginilor HTML folosind una din urmatoarele metode:

- Folosindu-ne de atributele HTML corespunzatoare unor evenimente
  - Specifice ferestrei de browser (window): onload, onunload, onresize, etc
  - Specifice formularelor HTML: onsubmit, onreset, onchange, onfocus, etc
  - Specifice lucrului cu tastatura: onkeydown, onkeypress, onkeyup
  - Specifice lucrului cu mouse-ul: onmousedown, onmouseup, onclick, ondblclick, oncontextmenu, onmouseover, onmouseout, etc
  - Evenimente pentru lucrul cu elemente drag & drop: ondragstart, ondrag, ondragend, etc
  - Evenimente pentru lucrul cu clipboard-ul: oncopy, oncut, onpaste
  - Evenimente pentru lucrul cu continut multimedia: oncanplay, onplay, onended, onerror, etc
  - Other events
  - La fel ca inline CSS: **ne-recomandat**, deoarece amestecam continut cu logica de interactiune!

Javascript - JavaScript si HTML

Putem implementa cod JavaScript in cadrul paginilor HTML folosind una din urmatoarele metode:

- Folosindu-ne de tag-ul <script> si incluzand codul JavaScript in cadrul acestui tag
  - Atribute optionale: “async” (executie asincrona fata de restul paginii), “defer” (executie dupa finalizarea parsarii paginii). In absenta acestor atribute, executia este imediata, lucru ne-dorit in mai multe situatii!
  - Atentie la <script>-uri injectate dinamic (ex: din codul de back-end) si care pot fi compuse pe baza user input-ului si/sau a unor fisiere uploadate de user (XSS)

Javascript - JavaScript si HTML

Putem implementa cod JavaScript in cadrul paginilor HTML folosind una din urmatoarele metode:

- Folosindu-ne de tag-ul <script> si mentionand o sursa externa prin atributul “src”
  - Script-urile externe pot fi script-uri aflate in foldere / sub-foldere ale website-ului nostru (caz pentru care se aplica regulile de path discutate deja), sau pot fi script-uri servite de alte site-uri (ex: CDN - vezi script-urile JavaScript servite de BootStrap)
  - Atributele “async” si “defer” pot fi folosite si in acest caz

Javascript - Evenimente

Pentru a facilita o experienta dinamica de utilizare a aplicatiilor web, standardul HTML defineste un set de evenimente, pe care programatorii pot decide sa le “intercepteze” si sa ruleze o anumita logica / functionalitate, continute intr-un bloc de cod JavaScript (de regula o functie);

Atasarea unei functionalitati pentru tratarea unui eveniment se poate realiza:

- Prin utilizarea atributelor HTML (ne-recomandat)

```
<button id="btn" onclick="alert('You clicked me!');">Button</button>
```

- Din codul javascript: existand mai multe variante si diferente intre browsere (in special intre IE si restul lumii), insa varianta moderna & recomandata (si suportata si de ultimele versiuni de IE) fiind prin utilizarea functiei addEventListener

Javascript - Evenimente

Evenimentele declansate de browser poarta cu ele diverse informatii care pot fi de interes pentru programatori (ex: cine este sursa evenimentului, care este tasta apasata, care este coordonata mouse-ului, etc). Aceste informatii vor fi transmise in mod automat (daca mecanismul de tratare folosit nu este de asa natura facut incat sa nu tina cont de ele) sub forma unui parametru, denumit de regula obiectul eveniment (the event object).

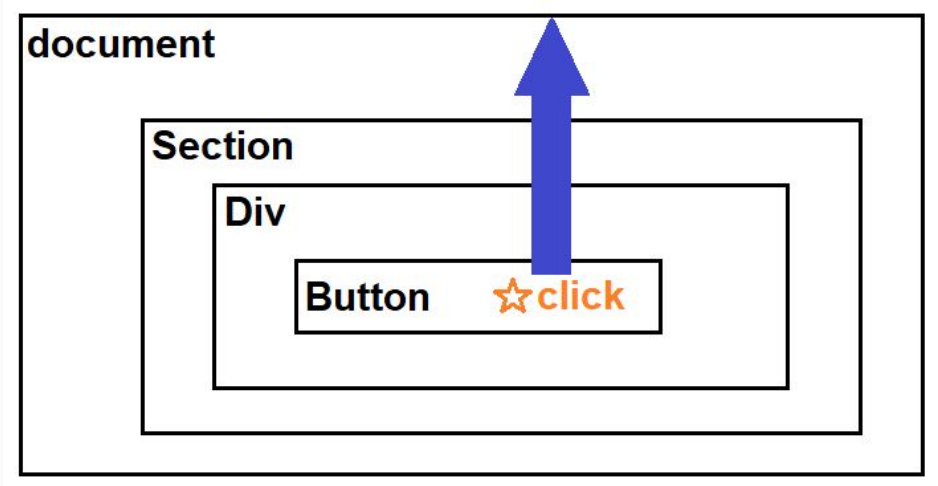
Javascript - Evenimente

In functie de modul in care browserele declanseaza evenimentele si modul in care acestea pot fi tratate de codul JavaScript, deosebim urmatoarele tipuri de gestionare e evenimentelor:

- Event Bubbling
- Event Capturing
- Mixt (prima data Event Capturing, apoi Event Bubbling)

Javascript - Evenimente - Event Bubbling

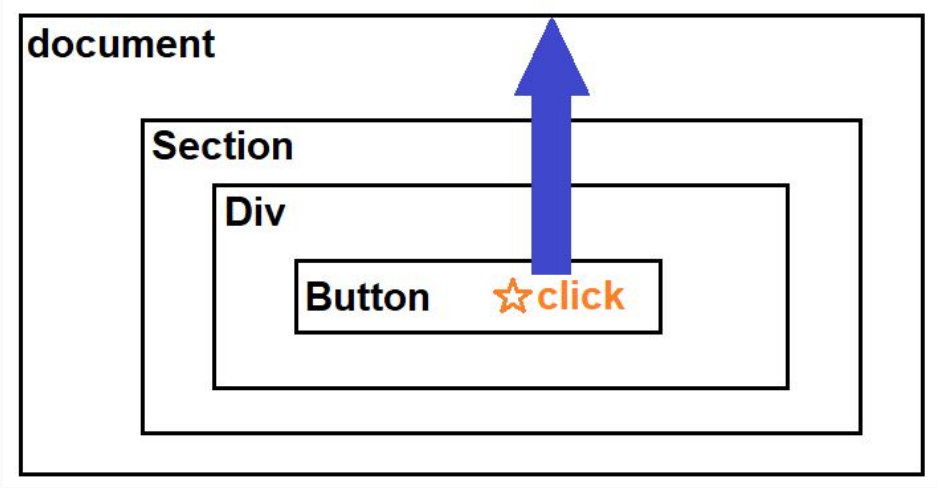
Event Bubbling: evenimentul este declansat de catre elementul DOM cel mai apropiat de locul producerii evenimentului (ex: butonul pe care s-a apasat click), iar apoi evenimentul se propaga (bubble up) spre parintele elementului, apoi spre parintele parintelui, s.a.m.d, pana se ajunge la elementul radacina (document).



Javascript - Evenimente - Event Bubbling

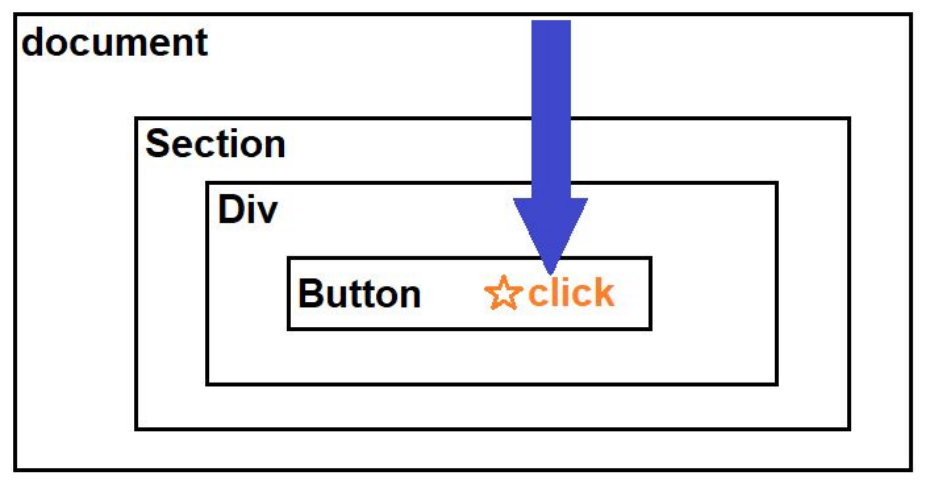
In cursul propagarii de la sursa catre elementele parinte pana la radacina, daca exista mai multe elemente care au atasate un handler (functie JavaScript pentru tratarea evenimentului), toate aceste handlere vor fi apelate!

Pentru a opri propagarea la nivelul unui anumit handler, se poate utiliza metoda stopPropagation() a obiectului eveniment.



Javascript - Evenimente - Event Capturing

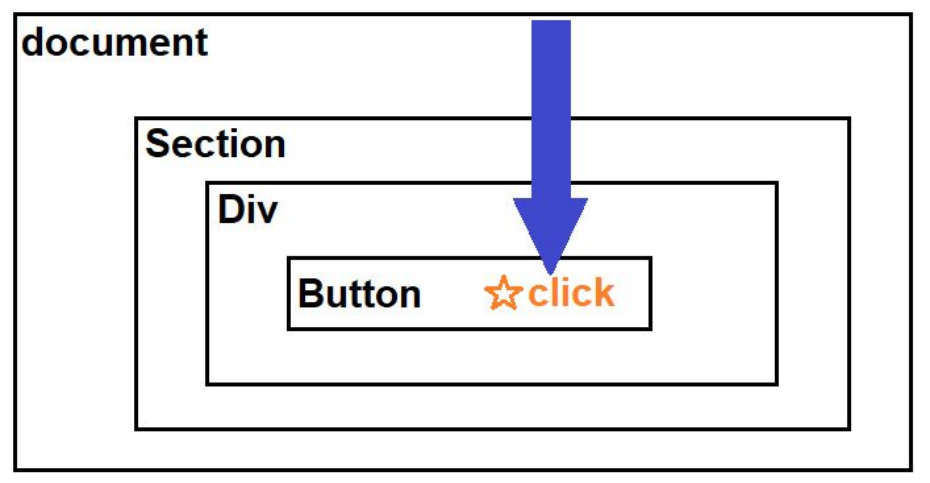
Event Capturing: evenimentul este declansat de catre elementul DOM radacina (document), iar apoi evenimentul se propaga (capture down) pe ierarhia de descendeti spre pana se ajunge la elementul cel mai apropiat de locul producerii evenimentului (ex: butonul pe care s-a facut click).



Javascript - Evenimente - Event Capturing

Idem, si pentru Event Capturing, daca de-a lungul propagarii evenimentului, exista mai multe handlere atasate, toate aceste handlere se vor executa.

Si pentru Event Capturing propagarea poate fi oprita folosind functia stopPropagation()



Javascript - Evenimente
Event bubbling - vs - Event capturing

In mod traditional, Internet Explorer suporta doar prindere si tratare de evenimente in stil “Event bubbling”, iar restul browserelor (conform cu specificatia W3C / DOM) defineau o tratare in 3 faze:

- In prima faza avea loc propagarea evenimentului in stil “Event Capturing” cu executarea handlerelor atasate (daca existau), pana se ajungea la elementul sursa
- Apoi se executa handler-ul atasat elementului sursa
- Apoi urma propagarea in stil “Event Bubbling” cu executarea handlere-lor atasate (daca existau), pana se ajungea la elementul sursa
- De asemenea, atunci cand se atasa un handler, dezvoltatorul era obligat sa specifice pentru care faza (capture - vs - bubble) ataseaza acel handler, asa incat sa se evite dubla executie

Javascript - Evenimente
Event bubbling - vs - Event capturing

Incepand de la versiunea 9+ si IE s-a aliniat standardelor W3C / DOM, insa din cauza “istoriei” sale vis-a-vis de tratarea evenimentelor, web developerii s-au obisnuit sa ataseze de regula handlere pentru faza de event bubbling (pentru ca aplicatiile sa functioneze consistent, indiferent de browser).

Specificarea fazei pentru care se face atasarea evenimentului se face prin parametrul 3 (boolean “useCapture”) al functiei addEventListener(). In lipsa specificarii acestuia (sau pentru alte stiluri de atasare), handler-ul va fi atasat pentru faza de event bubbling.

Javascript - Framework-uri Javascript

Pentru a oferi o interfata de programare (API) consistenta, care sa rezolve diferentele care exista intre browsere vis-a-vis de implementarea standardelor web si pentru a oferi un set bogat de optiuni si un plus de productivitate, au fost dezvoltate de-a lungul timpului mai multe framework-uri (librarii) javascript:

- Pentru dezvoltarea de aplicatii SPA:
  - React (Facebook)
  - Angular (Google)
  - Vue
- Alte mentiuni:
  - jQuery - o librarie mica, axata in jurul operatiunilor de manipulare a DOM-ului si a lucrului cu evenimente si apeluri HTTP / Ajax.